

Programmer's Guide

AT&T Personal Computer 6300 GWBASIC By Microsoft® Written by Agora Resource, Inc. Lexington, MA

©1984, 1985 AT&T ©1983, 1984 By Microsoft® All Rights Reserved Printed in USA

NOTICE

The information in this document is subject to change without notice. AT&T assumes no responsibility for any errors that may appear in this document.

Contents

1	Introduction Introduction Major Features	1-2 1-3
	Syntax Conventions	1-4
	Line Format Character Set	1-7
	Reserved Words	1-9 1-10
0	Getting Started	
2	Initialization Procedure	2-2
	Modes of Operation	2-3
	Keyboard	2-4
	The GWBASIC Screen Editor	2-12
	 Using Your System as a Calculator 	2-23
	Entering a Program	2-26
	Listing a Program	2-29
	Saving a Program	2-30
	Loading a Program	2-31
	Executing a Program Program Interrupts	2-32 2-37
	Frogram interrupts	2-37
	Variable Types	
4	Constants	3-2
_ 7	Variables	3-7
	Expressions and Operators	3-14

How MS-DOS Keeps Track of Your Files File Specification Commands for Program Files Disk Data Files — Sequential and Random Access Graphics Selecting the Screen Attributes Text Mode Graphics Mode Asynchronous Communications Opening Communications Files Communication I/O Communication I/O Functions Command References Introduction Commands, Statements, and Functions with			
Selecting the Screen Attributes Text Mode Graphics Mode Asynchronous Communications Opening Communications Files Communication I/O Communication I/O Functions Communication Command References Introduction Commands, Statements, and Functions with	4	Device Independent Input/Output How MS-DOS Keeps Track of Your Files File Specification Commands for Program Files Disk Data Files — Sequential	4-2 4-3 4-5 4-18
Text Mode Graphics Mode Asynchronous Communications Opening Communications Files Communication I/O Communication I/O Functions Communication Command References Introduction Commands, Statements, and Functions with	<u> </u>	Graphics	
Communications Opening Communications Files Communication I/O Communication I/O Functions Command References Introduction Commands, Statements, and Functions with	O	Text Mode	5-2 5-4 5-7
Communication I/O Communication I/O Functions Command References Introduction Commands, Statements, and Functions with	G		
Introduction Commands, Statements, and Functions with	U	Communication I/O	6-2 6-3 6-4
Commands, Statements, and Functions with	7	Command References	
		Commands, Statements,	7-2
Examples		Examples	7-16

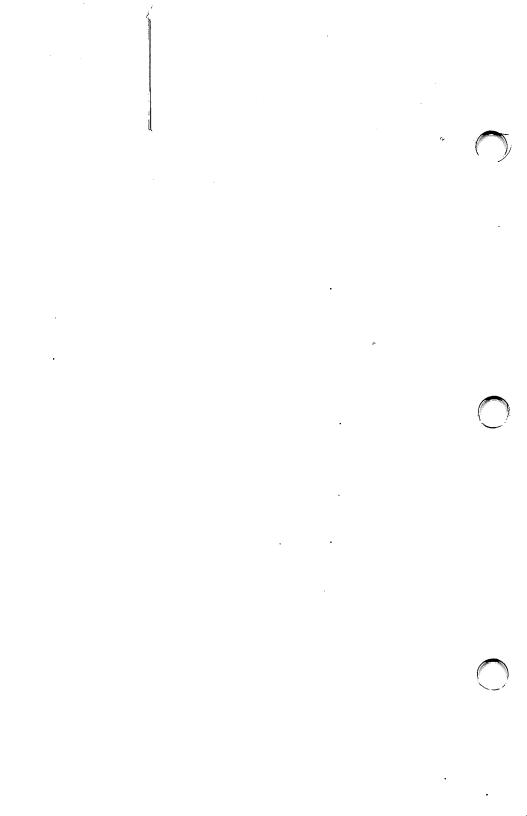
Appendices

A	Tables Hexadecimal Conversion Tables ASCII Codes Extended Codes Hexadecimal to Decimal Conversion Tables Derived Functions	A-3 A-4 A-8 A-10 A-12
В	Advanced Features Memory Allocation Internal Representation Calling Subroutines Event Trapping	B-2 B-4 B-6 B-22
C	Conversion of Programs to GWBASIC Introduction String Dimensioning MAT Functions Multiple Assignments Multiple Statements PEEKs and POKEs IFTHEN[ELSE] File I/O Graphics Sounding the Bell	C-2 C-3 C-6 C-7 C-8 C-9 C-10 C-11 C-12
D	Error Codes and Error Messages Error Messages Error Codes	D-2 D-4

Glossary

Index

Supplement: Display Enhancement Board



Introduction

1

- Introduction
- Major Features
- Syntax Conventions
- Line Format
- Character Set
- Reserved Words

INTRODUCTION

GWBASIC is the most extensive implementation of BASIC available for personal computers. It meets the requirements of the ANSI standard for BASIC, and supports many features rarely found in other BASICs. It provides sophisticated string handling, structured programming features, and improved graphics.

GWBASIC gives you ease of use plus features that make your personal computer perform at its best.

UNIX is a trademark of AT&T Bell Laboratories.

MS[™]-DOS is a trademark of Microsoft Corporation.

Microsoft® is a registered trademark of Microsoft Corporation.

MAJOR FEATURES

Some of the special features of GWBASIC are:

- UNIX[™] style MS[™]-DOS interface for a userfriendly operating environment
- Re-directable standard input and output
- Device communication commands to initialize and communicate with peripheral devices
- Tree-structured disk directories
- Improved Disk I/O facilities for large files
- Advanced screen editing
- Enhanced Graphics commands
- User-defined Keyboard, Error, and Event Trapping
- Precise error reporting with ERDEV and ERDEV\$
- Optional double precision transcendentals
- Precise control of memory allocation
- CALL statements with parameter passing
- Chaining with common variables to programs larger than the available memory
- Optional declaration of variable names

SYNTAX CONVENTIONS

Uppercase letters and words, and the symbols listed below, should be typed in the actual line exactly as shown.

$$(), ; : = / # \$ - > <$$

In the statement:

WRITE # filenum, list-of-expressions

and the comma (,) after filenum should be typed as shown.

 Lowercase letters and words represent variable information (or parameters) that the user must provide. In the statement:

KILL filespec

filespec should be replaced by a specific value—for example, "MYFILE".

 The symbols listed below are used to define the syntax of a line, but should not be typed in the actual line:

vertical stroke indicates alternatives
braces indicate a choice
brackets indicate options
ellipsis indicates repetition
underscore joins parts of names in a multiple-

word parameter

• Braces group related items (divided by a vertical stroke), such as alternatives.

$\{A|B|C\}$

indicates that you must choose one of the items enclosed within the braces.

A or B or C

 Brackets also group related items (divided by a vertical stroke); however, everything within the brackets is optional and may be omitted.

[A|B|C]

indicates that you may choose one of the items enclosed within the brackets or that you may omit all of the items.

 An ellipsis indicates that the preceding item or group of items may be repeated more than once in succession.

A [,B]...

indicates that ${\bf A}$ can be typed alone or can be followed by

.B

once or more in succession.

Note

A [,list_of_B]

is also permitted and has the same meaning as

A [,B]...

 The underscore character (__) can be used to join names in a multiple-word parameter. For example:

ENVIRON\$ (nth_parm)

Characters which appear in a listing in **bold** face represent characters entered through the
 keyboard.

LINE FORMAT

GWBASIC lines may contain a maximum of 255 characters and have the following format:

[nnnnn] statement [:statement]...['comment] CR

A GWBASIC program line always begins with a line number (an unsigned integer in the range 0 to 65,529), and ends with a carriage return (CR). A program line is stored in memory as soon as you enter CR.

A GWBASIC immediate line, i.e., a line that is executed as soon as you enter it, always begins with a letter, as you have to omit the line number in this case.

More than one GWBASIC statement may be placed on a line, but each successive statement must be separated from the last by a colon.

At the end of a GWBASIC line (before CR) you may enter a comment string preceded by a single quotation mark (').

A comment string preceded either by the keyword REM or by a single quotation mark may also be written just after the line number.

You can extend a logical line over more than one physical line by pressing CTRL-CR or by continuing typing and letting the logical line wrap around to the next physical line.

All GWBASIC lines shown in this manual end with CR unless specifically stated otherwise.

Examples:

10 FOR K = 1 TO 20

is a GWBASIC program line.

100 GOSUB 1000 'branch to SUB1

is a GWBASIC program line with a comment at the end.

1000 'SUB1

is a GWBASIC program line which contains only a comment.

PRINT AS

is a GWBASIC immediate line.

 Every GWBASIC program line begins with a line number. Line numbers indicate the order in which the program lines are stored in memory. Line numbers are also used as references in branching and editing.

For the EDIT, LIST, AUTO, and DELETE commands, a period (.) may be used to reference the current line.

CHARACTER SET

GWBASIC recognizes upper and lower case letters of the alphabet, the digits 0 through 9, and the following special characters:

Rlank

- = Equals sign or assignment symbol
- + Plus sign
- Minus sign
- * Asterisk or multiplication symbol
- / Slash or division symbol
- Up arrow or exponentiation symbol
- (Left parenthesis
-) Right parenthesis
- % Percent sign or integer type declaration character
- # Number (or pound) sign or double precision type declaration
- S Dollar sign or string type declaration character
- ! Exclamation point or single precision type declaration character
- Left bracket (*)
- l Right bracket (*)
- , Comma
- . Period or decimal point
- Single quotation mark (apostrophe)
- " Double quotation mark (string delimiter)
- : Semicolon
- : Colon & Ampersand
- ? Question mark (PRINT abbreviation)
- < Less than
- > Greater than
- \ Backslash or integer division symbol
- @ At sign
- __ Underscore (*)
- Vertical line or pipe
- { Left brace
- Right brace
- Grave accent
- ~ Tilde
 - (*) Since these symbols are not used as operators in the language, they may be used to define the syntax (see Syntax Conventions above). They should be typed in the actual line only if they belong to a string constant.

RESERVED WORDS

GWBASIC comprises a set of statements, commands, function names, and operator names which are treated as reserved words, and which cannot be used as variable names. The total list of GWBASIC reserved words is as follows:

ABS	EQV	LSET
AND	ERASE	MERGE
ASC	ERDEV	MID\$
ATN	ERDEV\$	MKDIR
AUTO	ERL	MKD\$
BEEP	ERR	MKI\$
BLOAD	ERROR	MKS\$
BSAVE	EXP	MOD
CALL	FIELD	NAME
CALLS	FILES	NEW
CHAIN	FN	NEXT
CHDIR	FIX	NOT
CHR\$	FOR	OCT\$
CINT	FRE	OFF
CIRCLE	GET	ON
CLEAR	GOSUB	OPEN
CLOSE	GOTO	OPTION
CLS	HEX\$	OR
COLOR	IF	OUT
COM	IMP	PAINT
COMMON	INKEY\$	PEEK
CONT	INP	PLAY
COS	INPUT	PMAP
CSNG	INPUT#	POINT
CSRLIN	INPUT\$	POKE
CVD	INSTR	POS
CVI	INT	PRESET
CVS	IOCTL	PRINT
DATA	IOCTL\$	PRINT#
DATE\$	KEY	PSET
DEF	KILL	PUT
DEFDBL	LEFT\$	RANDOMIZE
DEFINT	LEN	READ
DEFSNG	LET	REM
DEFSTR	LINE	RENUM
DELETE	LIST	RESET
DIM	LLIST	RESTORE
DRAW	LOAD	RESUME
EDIT	LOC	RETURN
ELSE	LOCATE	RIGHT\$
END	LOF	RMDIR
ENVIRON	LOG	RND
ENVIRON\$	LPOS .	RSET
EOF	LPRINT	•

RUN	STRING	USR
SAVE	STRING\$	VAL
SCREEN	SWAP	VARPTR
SGN	SYSTEM	VARPTR\$
SHELL	TAB	VIEW
SIN	TAN	WAIT
SOUND	THEN	WEND
SPACE\$	TIMER	WHILE
SPC	TIME\$	WIDTH
SQR	TO	WINDOW
STEP	TROFF	WRITE
STICK		
STOP	TRON	WRITE#
STR\$	USING	XOR

the second second

2

Getting Started

- Initialization Procedure
- Modes of Operation
- Keyboard
- The GWBASIC Screen Editor
- Using Your System as a Calculator
- Entering a Program
- Listing a Program
- Saving a Program
- Loading a Program
- Executing a Program
- Program Interrupts

INITIALIZATION PROCEDURE

To start GWBASIC, the MS-DOS operating system must first be installed. When MS-DOS has been installed and the system prompt:

A >

is displayed, enter the GWBASIC command:

GWBASIC

to load GWBASIC from the diskette inserted in drive A into memory.

Upon loading, GWBASIC responds with a screen similar to the one shown below.

GWBASIC 2.02 (C) Copyright Microsoft 1983, 1984 AT&T Personal Computer Release 1.1 Copyright (C) 1984, 1985 by AT&T, all rights reserved

XXXXX Bytes Free Ok

- Insert a diskette containing your GWBASIC programs and execute a program, or
- enter GWBASIC program or immediate lines.
- To exit from GWBASIC and return to MS-DOS, enter:

SYSTEM

This closes all data files before returning to MS-DOS. Your GWBASIC program is no longer in memory. MS-DOS remains resident.

MODES OF OPERATION

The GWBASIC Interpreter may be used in either of two modes: direct mode or indirect mode.

• In direct mode, statements and commands are executed as they are entered. They are not preceded by line numbers. After each direct statement followed by a carriage return, the screen will display the "Ok" prompt. Results of arithmetic and logical operations may be displayed immediately and stored for later use, but the instructions themselves are lost after execution. Direct mode is useful for debugging and for using the GWBASIC Interpreter as a calculator for quick computations that do not require a complete program.

Example

Ok PRINT 45+3 48 Ok

Indirect mode is used for entering programs. Program lines are preceded by line numbers and are stored in memory. The program stored in memory is executed by entering the RUN command.

Example

Ok 10 PRINT 45+3 RUN 48 Ok

KEYBOARD

The Keyboard is divided into three sections:

- Ten function keys, named F1 through F10 on the left-hand side of the keyboard.
- The standard typewriter keyboard in the center, used to enter letters, numbers, special characters and control characters.
- The numeric keypad on the right-hand side of the keyboard, used to enter numbers, numeric operators, and the Screen Editor commands.

FUNCTION KEYS

There are 10 function keys on the keyboard.

These function keys can be tailored to the user's needs using the KEY and ON KEY statements.

The KEY statement can be used to assign a specific command or sequence of characters to a function key, other than the pre-assigned standard commands. The ON KEY statement can be used to generate program interrupts via a specified function key.

Refer to the Reference section for further details.

TYPEWRITER KEYBOARD

The standard typewriter keyboard is used to enter letters, numbers, special characters, and control characters.

Shift Keys

If you want to enter upper case letters or the upper symbol on those keys containing two symbols, hold down one of the two \(\tau\) keys and press the corresponding key.

From now on we shall always refer to the \u00e1 keys as SHIFT keys by convention.

Carriage Return Key

The Carriage Return key is identified by the symbol ← □

By convention we shall refer to this key as the CR key.

You must press CR to close a GWBASIC line and send it to the system for processing.

Shift Lock for Letters

You can enable or disable Shift Lock for letters (A-Z) by pressing CAPS LOCK.

The CAPS LOCK key is similar to a typewriter Shift Lock Key, but it only gives you uppercase letters, and will not give you the upper symbols on the numeric or other keys.

Backspace

The backspace key ← moves the cursor one position to the left, erasing the last character you have typed.

To move the cursor to the left without erasing any characters, you should use the Cursor Left Key located on the numeric keypad.

Control Characters

You can generate control characters by holding down the CTRL or ALT key while pressing another key. GWBASIC recognizes a number of control characters.

CTRL-BREAK

- 1. To interrupt the program at the following GWBASIC instruction and return to GWBASIC Command Level.
- 2. To cancel automatic line numbering mode while entering a program.
- 3. To return to Command Level, without saving any changes that you made to the current line.

CTRL-G

Sounds the bell.

CTRL-NUM LOCK

Causes the system to 'pause' so as to temporarily halt printing or program listing. The pause continues until you press any key (except SHIFT, CTRL or ALT).

CTRL-T

Scrolls the function key display horizontally across the screen (on the 25th screen line), when the width is 40. When the width is 80, it toggles the Function Key display ON and OFF.

CTRL-ALT-DEL

Performs a System Reset by holding down the CTRL and ALT keys, and then pressing DEL.

CTRL-PRTSC

All text sent to the screen is also sent to the system printer. A second CTRL-PRTSC will stop printing. If you press PRTSC while holding down SHIFT, MS-DOS will make a single printed copy of the entire display screen.

CTRL-L

Outputs a formfeed character. It has the same function as the CLS statement, (i.e., it clears the screen or the current graphics viewport, if a viewport has been defined).

CTRL-Z

Sets an end of file condition (see the "OPEN COM Statement" in the Reference section).

Other control characters are described in the subsection entitled "Special Screen Editor Keys" later in this chapter.

Direct Entry of GWBASIC Keywords

You can type a GWBASIC Keyword by holding down the ALT key while pressing one of the alphabetic keys (A - Z). Keywords associated with each letter are listed below.

A - AUTO	N - NEXT
B - BSAVE	O - OPEN
C - COLOR	P - PRINT
D - DELETE	Q - ****
E - ELSE	R - RUN
F - FOR	S - SCREEN
G - GOTO	T - THEN
H - HEX\$	U - USING
I - INPUT	V - VAL
J - ****	W-WIDTH
K - KEY	X - XOR
L - LOCATE	Y - ****
M - MERGE	Z - ****
**** unused keys	

NUMERIC KEYPAD

A group of 15 keys at the right-hand side of the keyboard. It is arranged much like a standard calculator's keypad and is called "numeric keypad." It includes not only the numbers 0 through 9, the decimal point, the plus (+) and minus (-) keys, but also cursor movement keys, PGUP, PGDN, HOME, NUM LOCK, SCROLL LOCK, BREAK, END, INS, DEL, etc.

Note that some keys like SCROLL LOCK, PGUP, and PGDN are not used by GWBASIC, but they may be assigned meanings within a program.

Number Lock State

You can press the NUM LOCK key to shift the numeric keypad into upper-case. This mode provides the numbers 0 through 9 and the decimal point. (Holding down one of the two SHIFT keys produces the corresponding lower-case keys in this mode.) To return to lower-case, press NUM LOCK once again.

THE GWBASIC SCREEN EDITOR

All text entered while GWBASIC is at command level is processed by the GWBASIC Editor. This is a "screen line editor" which allows you to change a line anywhere on the screen (only one line at a time). Changes are only registered when you press CR on that line.

SPECIAL SCREEN EDITOR KEYS

The GWBASIC Editor recognizes 9 numeric Keypad Keys, the Backspace Key, and the CTRL Key to move the cursor, insert or delete characters.

The Keys and their functions are listed below.

HOME

Positions the cursor in the top left-hand corner of the screen.

CTRL-HOME

Clears the screen and moves the cursor to the "Home" position.

Moves the cursor up one line.

↓ Moves the cursor down one line.

Moves the cursor one position left. If the cursor is moved beyond the left edge of the screen, it appears at the right side of the screen on the preceding line.

Moves the cursor one position right. If the cursor is moved beyond the right edge of the screen, it appears at the left side of the screen on the following line.

CTRL →

Moves the cursor to the beginning of the following word, (i.e., to the next character to the right of the cursor in the set) [A..Z] or [a..z] or [0..9].

For example, in the following line:

30 IF L<=0 THEN 20

The cursor is under the letter L. If you press CTRL \rightarrow , the cursor will move to the beginning of the next word, which is 0:

30 IF L<=0 THEN 20

If you press CTRL → again, the cursor will move to the next word, which is THEN:

CTRL ←

Moves the cursor to beginning of the preceding word, (i.e., to the first character to the left of the cursor which is preceded by a blank or a special character). For example:

30 IF L<=0 THEN 20

The cursor is under the letter T. If you press CTRL \rightarrow the cursor will move to 0. Pressing CTRL \rightarrow again, it will move to L.

END

Moves the cursor from its current position to the end of the logical line. Subsequent characters are appended to the line.

CTRL END

Erases from the current cursor position to the end of the logical line, (i.e., until the carriage return is found).

INS

Switches into or out of Insert Mode. If Insert Mode is off (Overwrite Mode on), then it turns it on. If Insert Mode is on, then it turns it off (sets Overwrite Mode). The Insert Mode cursor is a half-height blinking block (in Text Mode) and is a blinking triangle to the left of the character (in Graphics Mode).

Overwrite mode is indicated by a different cursor, which is a slow-blinking under line. In Insert Mode, the characters immediately above, together with those following the cursor, move to the right as characters are inserted at the current cursor position. Line folding is observed; that is, as characters disappear off the right side of the screen, they return on the left on the following line.

When in Overwrite Mode, characters typed will replace existing characters on the line.

Insert Mode is turned off when you press the INS key again, or if you press any of the cursor movement keys, or CR.

TAB

When out of Insert Mode, pressing **TAB** moves the cursor over characters until the next tab stop is reached. Tab stops are set at every 8 character positions; that is, at positions 1, 9, 17, etc. For example, suppose we have the line:

20 INPUT "Length"; L

If you press the TAB key, the cursor will move to the 17th position as shown:

20 INPUT "Length"; L

When in Insert Mode, pressing TAB causes blanks to be inserted from the current cursor position to the next tab stop. Line folding is observed as explained under INS. For example, suppose we have the line:

20 INPUT "Length"; L

Blanks are inserted up to the 17th position by pressing the INS key and then the TAB key.

20 INPUT " Length"; L

DEL

Deletes the character at the current cursor position. All characters which follow the deleted character shift one position left. If a logical line extends beyond one physical line, characters on subsequent lines shift left one position to fill in the previous space, and the character in the first column of each subsequent line moves up to the end of the preceding line.

BACKSPACE

Causes the last character typed to be deleted, (i.e., on the character to the left of the cursor). All characters to the right of the deleted character shift left one position. Subsequent characters and lines within the current logical line move up as with the DEL key.

CTRL CR LINE FEED

Causes subsequent text to start automatically on the next screen line.

ESC DELETE LINE

The entire logical line containing the cursor is cleared. The line is not entered for processing. If it is a program line, it is not erased from the program in memory.

CTRL BREAK

Returns to Command Level, without saving any modifications that were made to the current line being edited. Unlike ESC, it does not erase the line from the screen.

CORRECTING THE CURRENT LINE

All text entered at GWBASIC Command Level is processed by the Screen Editor. You can therefore use any of the Special Screen Editor Keys.

GWBASIC remains at Command Level after the prompt Ok and until a RUN command is received.

Character Modification

If you make a mistake while entering a line then proceed as follows:

1 You discover the error. For example, suppose you have typed:

RUN "K.PROGR_

when you should have entered

RUN "A:PROGR_

2 Use Cursor-Left, or other cursor movement keys, to move the cursor to the appropriate position:

RUN "K, PROGR

3 Type the correct characters over the wrong ones:

RUN "A:PROGR

4 Move the cursor to the end of the line using Cursor Right or END keys:

RUN "A:PROGR

5 Continue typing if the line is not finished:

RUN "A:PROGRAM11"_

6 Enter CR to pass the line to GWBASIC. In this case the specified program is loaded from the diskette inserted in drive A and run.

Character Insertion

If you accidentally omit characters in the line you are entering, then proceed as follows:

1 You notice the error. Suppose you entered:

instead of:

2 Use Cursor-Left, or other cursor movement keys, to move the cursor to the appropriate position:

3 Press INS and type the letter R:

10 FOR■K=1 TO Note that, entering Insert Mode, the cursor becomes a halfheight block.

4 Press INS again to return to Overwrite Mode and Cursor-Right or END to move the cursor to the end of the line:
10 FOR K=1 TO__

Character Deletion

If you accidentally type an extra character in the line you are entering, then proceed as follows:

1	You discover the error. For example, suppose you typed:
	GOTTO_
	instead of:
	GOTO_
2	To erase the extra T, press Cursor Left, or other cursor movement keys, to move the cursor to the appropriate position:
	GOTTO
3	Press DEL:
	GOT <u>O</u>
4	Move the cursor using Cursor Right:
	GOTO_
5	Continue typing
	GOTO 1000_

Deleting Part of a Line

To erase a line from the current cursor position, press CTRL END.

Deleting an Entire Line

To cancel the line you are entering, press ESC anywhere in the line. It is not necessary to press CR.

MODIFYING PROGRAM LINES

Any line of text beginning with a number (0 to 65529) is considered to be a 'program line'. Suppose you have entered a program, i.e., a sequence of program lines, that you want to modify:

- To add a new line to your program, enter a valid line number followed by at least one nonblank character, followed by CR.
- To replace an existing line, enter a line number that matches an existing one, followed by the contents of the new line. The new line will replace the existing one.
- To delete a line enter a line with the same line number as the line to be deleted, followed by CR. An "Undefined line number" error is returned if an attempt is made to delete a line which does not exist.

Note: **ESC** should not be used to delete program lines, since this erases from the screen only, and not from the program in memory.

- To delete a group of lines, enter a DELETE command indicating the range of lines to be deleted (see the Reference Section).
- To delete the program resident in memory, enter a NEW command (see the Reference Section).
- To modify a program line which is already displayed on the screen, move the cursor to the appropriate position (by the cursor movement keys); modify the line using any of the techniques described above to change, delete, or insert characters to the line; press CR to pass the modified line to GWBASIC.

• To modify a program line which is not displayed on the screen, use the EDIT command (see the Reference Section) to display the line, or the LIST command (see the Reference Section) to display a group of lines including the line you want to modify, move the cursor to the appropriate position, modify the line, and press CR.

Note: You can edit any line as long as it is visible on the screen. Once an immediate line has been sent to the system pressing CR, there is no way to edit it; this is not the case with program lines, as they may always be recalled for editing to the screen.

Remarks

No modifications are made within the program until CR is entered. It is sometimes more practical to move around the screen making corrections to several lines and then return to the first line changed and strike CR at the beginning of each line, thereby storing the modified lines in the program.

It is not necessary to move the cursor to the end of the logical line before typing the carriage return. The Screen Editor remembers where each logical line ends and transfers the whole line even if the carriage return is typed at the beginning of the line.

The preceding modifications only change the program in memory. In order to save these modifications permanently, use the SAVE command before entering a NEW command or leaving GWBASIC (see the SAVE and NEW commands in the Reference Section).

USING YOUR SYSTEM AS A CALCULATOR

You can use your Personal Computer as a calculator for quick computation and for debugging purposes.

When you are in GWBASIC, and the Ok prompt is on the screen, you can enter PRINT (or simply?), followed by any expression, and CR. The expression is evaluated and its value displayed. You can also enter LET, followed by any variable name, the assignment operator (=), any expression and CR. The value is assigned to the specified variable. You can use the variable to represent that value in successive computations. The keyword LET is optional; you can begin the line simply using the variable name.

CALCULATOR EXAMPLES

PRINT 3

The constant 3 is displayed.

PRINT 2+3

The expression 2+3 is evaluated, and its value (5) is displayed.

LET A=15.21

The constant 15.21 is assigned to the variable A. You can use A in successive computations to represent this value.

?A-1

The expression A-1 is evaluated, and its value (14.21) is displayed.

Note: ? is equivalent to PRINT

B = 2.3

The constant 2.3 is assigned to the variable B. The keyword LET is optional; you may begin with a variable name.

?A*B

The expression A*B is evaluated. Its value (34.983) is displayed.

?A*B-40

The expression A*B-40 is evaluated, and its value (-5.017002) is displayed.

Note: If a value is negative, the minus sign is displayed; if a value is positive, no sign is displayed.

ENTERING A PROGRAM

A GWBASIC program consists of a series of statements. A statement is a complete instruction in GWBASIC, telling your computer to perform specific operations.

You can enter either one or several statements per line. In the latter case, each statement must be separated from the last by a colon (:).

Each line in a GWBASIC program begins with a line number: an integer greater than or equal to 0 and less than or equal to 65529. The line ends when you press CR.

A GWBASIC line may contain a maximum of 255 characters including the carriage return. Any extra characters will be truncated when you enter CR.

When you are in GWBASIC, and the Ok prompt is on the screen, you can enter a program.

Example

Enter:

NEW

This clears memory.

Then enter:

```
10 REM RECTANGLE1
20 INPUT "Length";L
30 IF L<=0 THEN 20
40 INPUT "Width";W
50 IF W<=0 THEN 40
60 LET AREA=L*W
70 PRINT "Area=";AREA;"L=";L;" W=";W
80 GOTO 20
90 END
```

It is conventional to use an interval of 10 between each line number. This allows you to modify the program simply by inserting statements between existing lines.

The above statements form a complete program that solves a very simple problem. The problem is to find the area of a rectangle by entering the values of length and width via the keyboard. It has been selected both for its simplicity and to illustrate a variety of GWBASIC features. Other more concise solutions exist.

AUTOMATIC LINE NUMBERING

You can use the AUTO command (see the Reference Section), to generate a line number automatically each time you press CR by pressing CTRL BREAK.

LISTING A PROGRAM

Once a program is in main memory it can be displayed or listed. To list your program, enter either the LIST command (the listing will appear on the screen) or, if a printer is connected, the LLIST command (the listing will be printed out).

The LIST and LLIST commands edit your program by converting to upper case letters any keywords, variable names, and function names and to PRINT any question mark (?) used instead of PRINT. Statements are ordered in ascending line number sequence, even though you may have entered them in a different order.

To list our sample program on the screen enter:

LIST

The screen display:

```
10 REM RECTANGLE1
20 INPUT "Length";L
30 IF L<=0 THEN 20
40 INPUT "Width";W
50 IF W<=0 THEN 40
60 LET AREA=L*W
70 PRINT "Area=";AREA;" L=";L;" W=";W
80 GOTO 20
90 END
0k
```

Note that at the end of a listing your system enters command level and displays the Ok prompt; the program can now be edited as required.

SAVING A PROGRAM

A program is kept in memory as long as your computer is switched on and GWBASIC is running and LOAD is not executed. As soon as you turn off your computer, do a system reset, exit GWBASIC with a system command, or LOAD another program, your program is lost. If you want to retain your newly written program for future use, then you must enter a SAVE command to store the program on a disk.

You should save the current program (i.e., the program presently resident in the main memory) on the following occasions:

- before you turn the machine off or do a system reset
- before entering a new program from the keyboard
- before loading another program in from disk
- before returning to MS-DOS by entering a SYSTEM command
- to replace the old version of your program with one you've just edited

LOADING A PROGRAM

If the program you want to enter into the main memory resides on a disk, you must issue a LOAD command. LOAD deletes all variables and program lines currently residing in memory. Before entering a LOAD command save the current program if you want to use it again, unless you already have a copy.

To load a program file from a disk, you must specify the drive before the file name, unless the file resides on the default drive. For example:

LOAD "B:ROOT1"

Loads the program if ROOT1 resides on the diskette inserted in drive B.

If you specify the R option, all open data files are kept open and the program is run after it is LOADed. For example:

LOAD "B:ROOT1",R

If you do not specify the R option, LOAD closes any data files that may be open.

EXECUTING A PROGRAM

Once a program is in main memory, it can be executed (or "run", as this is frequently called). To tell your system to execute a program, you must enter a RUN command (or a LOAD with the option R).

The RUN command runs the current program, i.e., the program currently in memory, or loads a program from a disk and runs it (if you enter a file specifier after the keyword RUN). For example:

RUN "B:RECTANGLE1"

Note that a file specifier is a string expression or, in particular, a string constant. If it is a string constant as in the example above, it must be enclosed within quotation marks (").

If you specify the R option all open data files are kept open, thus you can re-use these files in the new program without having to open them again.

Before entering a RUN filename (or RUN filename, R), save your current program (unless you already have a copy).

GWBASIC statements are executed in line number sequence, unless a control statement (GOTO, ON...GOTO, IF...GOTO...ELSE, IF...THEN...ELSE, FOR/NEXT, WHILE/WEND) or a subroutine call statement (GOSUB, ON...GOSUB) dictates otherwise.

RUNNING A SAMPLE PROGRAM

Let us run our sample program. Let us suppose it is already in memory, entered through the keyboard or loaded from disk by the LOAD command.

Enter:

LIST

```
10 REM RECTANGLE1
20 INPUT "Length";L
30 IF L<=0 THEN 20
40 INPUT "Width";W
50 IF W<=0 THEN 40
60 LET AREA=L*W
70 PRINT "Area=";AREA;" L=";L;" W=";W
80 GOTO 20
90 END
0k
```

to check that this program is in main memory. The listing will appear on the screen. At the end of the listing, when Ok appears on the screen, enter:

RUN

Enter values for length and width in response to the program's prompts.

For example:

Length? 3.5

Width? 4.2

Area = 14.7 L= 3.5 W= 4.2

Length? -7.3

Length? 7.3

Width? 1.3Q

?Redo from start

Width? 1.32

Area= 9.636 L= 7.3 W= 1.32

Length? (Press CTRL and BREAK keys)

Break in 20

Ok

If you enter a negative value for W, statement 40 is executed again, as statement 50 returns control to statement Q for W) the system displays an error message:

?Redo from start

and you must re-enter the value. This program continues to run until you press CTRL BREAK to stop execution. Your system displays a "Break in nnnnn" message and returns to Command Level. To resume execution enter:

CONT

PROGRAM INTERRUPTS

Three types of program interrupts are possible:

- Manual interrupts
- Automatic interrupts
- Programmable interrupts

If you press CTRL BREAK, (manual interrupt), or a STOP, or an END statement is executed (programmed interrupt), then the program is interrupted, GWBASIC enters Command Level and displays OK. CTRL BREAK and STOP do not close any data file and display a "Break in nnnnn" message. END closes all data files and does not display a "Break in nnnnn" message.

In any case you can resume execution by entering a CONT command. You can display program variables (by immediate PRINT or PRINT USING statements) or change their values (by immediate LET or SWAP statements). You can also display program lines by an EDIT or LIST command, and modify them.

If you modify lines, you cannot continue execution via a CONT command. You can only rerun the program by entering RUN.

If a Syntax error is found (automatic interrupt), then the program is interrupted, GWBASIC displays the error message at the line that caused the error, positioning the cursor under the first digit of the line number.

You can modify the line, and then rerun the program by entering RUN. You cannot continue execution by entering CONT.

If you want to examine the contents of some variables before making any modifications you should press CTRL BREAK to return to Command Level. After examining the contents of the variables you can edit the line and rerun the program.

For example:

10 A=2\$6 RUN

?Syntax Error in 10

10 A=2\$6

If an error (other than a Syntax error) is found (automatic interrupt), the program is interrupted, GWBASIC displays the error message, enters Command Level and displays OK.

You can either display program variables or display program lines by an EDIT or LIST command, and then modify them.

You cannot continue execution by entering a CONT command, but you can rerun the program by entering RUN.

For example, running a program which contains:

100 FOR K=

will cause:

Missing operand in 100 OK

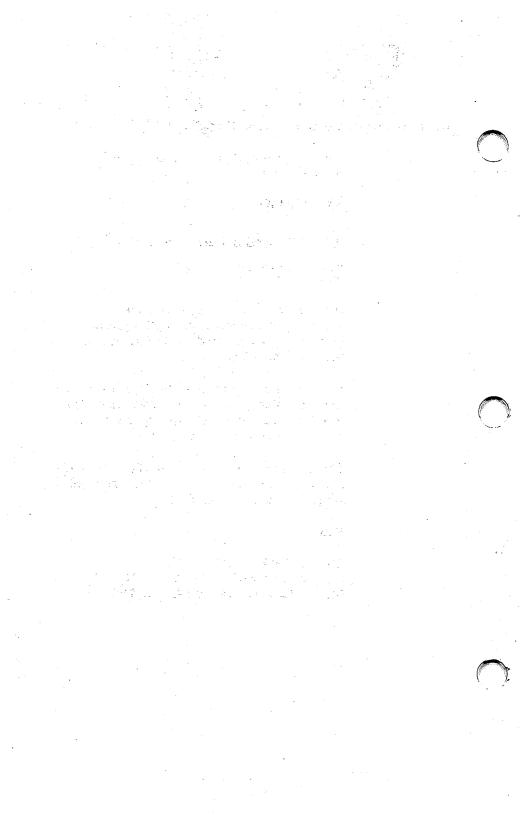
If an error occurs and the error trapping is enabled (programmed interrupt), program execution is transferred to the line specified by the ON ERROR statement.

An error trapping routine should check for all the particular errors that the user wishes to recover from, and should specify the course of action to be taken in each case.

This involves either correcting the error, and resuming execution at a specified statement or; returning to Command Level.

Example

10 ON ERROR GOTO 100 20 INPUT "WHAT IS YOUR BET";B 30 IF B>5000 THEN ERROR 200



3

Variable Types

- Constants
- Variables
- Expressions and Operators

CONSTANTS

Constants are the values that GWBASIC uses during program execution. There are two types of constants: string and numeric.

A string constant is a sequence of up to 255 alphanumeric characters enclosed in double quotation marks.

Examples:

"READY" "\$80"

"acceleration rate"

Numeric constants are positive or negative numbers. GWBASIC numeric constants cannot contain commas. There are five types of numeric constants:

- Integer constants
 Whole numbers between -32768 and 32767.
 Integer constants do not contain decimal points.
- Fixed-point constants
 Positive or negative real numbers, i.e.,
 numbers that contain decimal points.
- Floating-point constants

 Positive or negative numbers represented in exponential form (similar to scientific notation). A floating-point constant consists of an optionally signed integer or fixed-point number (the mantissa) followed by the letter E and an optionally signed integer (the exponent). The range for floating-point constants is 10^{-38} to 10^{+38} .

Examples:

235.988E-7 = .0000235988 2359E6 = 2359000000

(Double precision floating-point constants are denoted by the letter D instead of E. See later in this chapter.)

Hex constants
 Hexadecimal numbers denoted by the prefix
 &H.

Examples:

&H76 &H32F &HFFAA

• Octal constants
Octal numbers denoted by the prefix &O or &.

Examples:

&0347 &1234

SINGLE AND DOUBLE PRECISION FOR NUMERIC CONSTANTS

Numeric constants may be either single precision or double precision numbers. Single precision numeric constants are stored with 7 digits of precision, and printed with up to 6 digits of precision. Double precision numeric constants are stored with 17 digits of precision and printed with up to 16 digits.

A single precision constant is any numeric constant that has one of the following characteristics:

- Seven or fewer digits and a decimal point.
- Exponential form using E.
- A trailing exclamation point (!).

Examples

46.8 - 1.09E-06 3489.0 22.5!

A double precision constant is any numeric constant that has one of the following characteristics:

- Eight or more digits and a decimal point.
- Exponential form using D.
- A trailing number sign (#).

Examples:

345692811 - 1.09432D-06 3489.0# 7654321.1234

VARIABLES

Variables are names used to represent values used in a GWBASIC program. The value of a variable may be assigned explicitly by the programmer, or it may be assigned as the result of calculations in the program. Before a variable is assigned a value, its value is assumed to be zero.

VARIABLE NAMES AND DECLARATION CHARACTERS

GWBASIC variable names may be any length. Up to 40 characters are significant. Variable names can contain letters, numbers, and the decimal point. However, the first character must be a letter. Special type declaration characters are also allowed (see below).

A variable name may not be a reserved word, but embedded reserved words are allowed. Reserved words include all GWBASIC commands, statements, function names, and operator names. If a variable begins with FN, it is assumed to be a call to a user-defined function. Variables may represent either a numeric value or a string. String variable names are written with a dollar sign (\$) as the last character. For example:

A\$ = "SALES REPORT"

The dollar sign is a variable type declaration character; that is, it "declares" that the variable will represent a string.

Numeric variable names may declare integer, single precision, or double precision values. The type declaration characters for these variable names are as follows:

% Integer variable

! Single precision variable

Double precision variable

The default type for a numeric variable name is single precision. Examples of GWBASIC

variable names:

Pl# Declares a double precision value.

MINIMUM! Declares a single precision value.

LIMIT% Declares an integer value.

NS Declares a string value.

ABC Represents a single precision value.

There is a second method by which variable types may be declared. The GWBASIC statements DEFINT, DEFSTR, DEFSNG, and DEFDBL may be included in a program to declare the types for certain variable names. These statements are described in detail in the

Reference Section.

ARRAY VARIABLES

An array is a group or table of values referenced by the same variable name. Each element in an array is referenced by an array variable that is subscripted with an integer or an integer expression. An array variable name has as many subscripts as there are dimensions in the array. For example V(10) would reference a value in a one-dimension array, T(1,4) would reference a value in a two-dimension array, and so on. The maximum number of dimensions for an array is 255. The maximum number of elements per dimension is 32,767. Both these values are also limited by the memory size of your system.

Wherever a variable name can be entered in a GWBASIC program line, an array element can also be entered. From now on, when speaking of a variable we shall mean either a simple variable or an array element.

MEMORY REQUIREMENTS

The number of bytes required by strings, variables and arrays is listed below.

Variable Type	Bytes
Integer	2
Single Precision	4
Double Precision	8
Array Type	Bytes
Integer	2 per
	element
Single Precision	4 per
	element
Double Precision	8 per
	element

Strings

3 bytes overhead plus the present contents of the string.

TYPE CONVERSION

When necessary, GWBASIC will convert a numeric constant from one type to another. The following rules and examples should be observed.

• If a numeric constant of one type is set equal to a numeric variable of a different type, the number will be stored as the type declared in the variable name. (If a string variable is set equal to a numeric value or vice versa, a "Type mismatch" error occurs.)

Example:

10 A%=23.42 20 PRINT A% RUN 23 Ok

 During expression evaluation, all of the operands in an arithmetic or relational operation are converted to the same degree of precision, (i.e., that of the most precise operand). Also, the result of an arithmetic operation is returned to this degree of precision.

Examples:

10 D#=6#/7 20 PRINT D# RUN .8571428571428571

The arithmetic is performed in double precision and the result is returned in D# as a double precision value.

10 D=6#/7 20 PRINT D RUN .8571429 Ok

The arithmetic is performed in double precision and the result is returned to D (single precision variable), rounded, and printed as a single precision value.

Logical operators convert their operands to integers and return an integer result.
 Operands must be in the range -32768 to 32767 or an "Overflow" error occurs. A full description of Logical Operators follows later in this chapter.

 When a floating-point value is converted to an integer, the fractional portion is rounded.

Example:

10 C%=55.88 20 PRINT C% RUN 56 Ok

• If a double precision variable is assigned a single precision value, only the first seven digits (rounded) of the converted number will be valid. This is because only seven digits of accuracy were supplied with the single precision value. The absolute value of the difference between the printed double precision number and the original single precision value will be less than 6.3E-8 times the original single precision value.

Example:

10 A2.04 20 B# = A 30 PRINT A;B# RUN 2.04 2.039999961853027 Ok

EXPRESSIONS AND OPERATORS

An expression may be a string or numeric constant, a variable, or a combination of constants and variables with operators. An expression always produces a single value.

Operators perform mathematical or logical operations on values. The GWBASIC operators may be divided into four categories:

- Arithmetic
- Relational
- Logical
- Functional

Each category is described in the following subsections.

ARITHMETIC OPERATORS

The arithmetic operators, in order of precedence, are as follows:

Operator	Operation	Sample Expression
^	Exponentiation	X^Y
_	Negation	$-\mathbf{x}$
*	Multiplication	X*Y
1	Division	X/Y
\	Integer Division	X\Y
MOD	Modulus Arithmetic	X MOD Y
+	Addition	X+Y
_	Subtraction	X-Y

To change the order in which the operations are performed, use parentheses. Operations within parentheses are performed first. Within the parentheses, the usual order of operations is maintained.

Some sample algebraic expressions follow, together with their GWBASIC counterparts.

Algebraic Expression	GWBASIC Expression
X+2Y	X+2*Y
$\frac{X - \frac{Y}{Z}}{XY}$	X - Y/Z
$\frac{XI}{Z}$	(X*Y)/Z
$\frac{X+Y}{Z}$	(X+Y)/Z
$(X^2)^Y$	(X ^2) ^Y
$X^{Y^{Z}}$	X ^(Y ^Z)
X(-Y)	X*(-Y)

Note:

Two consecutive operators must be separated by parentheses, as shown in the X*(-Y) example.

INTEGER DIVISION AND MODULUS ARITHMETIC

Two additional operators are available in GWBASIC: integer division and modulus arithmetic.

Integer division is denoted by the backslash (). The operands are rounded to integers before the division is performed, and the quotient is truncated to an integer. The operands must be within the range -32768 to 32767.

Example:

x = 10\4 PRINT x 2 Ok

Integer division follows multiplication and floating-point division in order of precedence.

Modulus arithmetic is denoted by the operator MOD. Modulus arithmetic yields the integer value that is the remainder of an integer division.

Example:

PRINT 10.4 MOD 4 2 Ok PRINT 25.68 MOD 6.99 5 Ok

Modulus arithmetic follows integer division in order of precedence.

OVERFLOW

If, during the evaluation of an expression, division by zero is encountered, the "Division by zero" error message is displayed, machine infinity (the largest number that can be represented in floating-point format) with the sign of the numerator is supplied as the result of the division, and execution continues. If the evaluation of an exponentiation operator results in zero being raised to a negative power, the "Division by zero" error message again is displayed, positive machine infinity is supplied as the result of the exponentiation, and execution continues.

If overflow occurs, the "Overflow" error message is displayed, machine infinity with the algebraically correct sign is supplied as the result, and execution continues.

RELATIONAL OPERATORS

Relational operators are used to compare two values. The result of the comparison is either "true" (-1) or "false" (0). This result may then be used to make a decision regarding program flow. (See "IF" statements, in the Reference section).

The relational operators are:

Operator	Relation Tested	Example
=	Equality	X=Y
<> or ><	Inequality	X < > Y
<	Less than	X < Y
>	Greater than	X > Y
<= or $=<$	Less than or equal to	X < = Y
>= or =>	Greater than or equal to	X > = Y

(The equal sign is also used to assign a value to a variable. See "LET" Statement in the Reference Section.)

When arithmetic and relational operators are combined in one expression, the arithmetic operation is always performed first. For example, the expression

$$X + Y < (T-1)/Z$$

is true if the value of X plus Y is less than the value of T-1 divided by Z.

More examples:

320 IF SIN(X) < 0 GOTO 1000 400 IF I MOD J <> 0 THEN K=K+1

LOGICAL OPERATORS

NOT X

Logical operators perform tests on multiple relations, bit manipulation, or Boolean operations. The logical operator returns a result which is either "true" (not zero) or "false" (zero). In an expression, logical operations are performed after arithmetic and relational operations. The outcome of a logical operation is determined as shown below.

The operators are listed in order of precedence.

1	0		1	1	0
0	1		1	0	1
			0	1	1
			0	0	0
X	Y	X AND Y	X	Y	X EQV Y
1	1	1	1	1	1
1	0	0	1	0	0
0	1	0	0	1	0
0	0	0	0	0	1
X	Y	X OR Y	X	Y	X IMP Y
1	1	1	1	1	1
1	0	1	1	0	0
0	1	1	0	1	1
0	0	0	0	0	1

Just as the relational operators can be used to make decisions regarding program flow, logical operators can connect two or more relations and return a true or false value to be used in a subsequent decision (see "IF" statements in the Reference Section.)

Example

IF D<200 AND F<4 THEN 80 IF I>10 OR K<0 THEN 50 IF NOT P THEN 100

Logical operators work by converting their operands to 16-bit, signed, two's complement integers in the range -32768 to 32767. (If the operands are not in this range, an error results.) If both operands are supplied as 0 or -1, logical operators return 0 or -1. The given operation is performed on these integers bit-by-bit; i.e., each bit of the result is determined by the corresponding bits in the two operands.

Thus, it is possible to use logical operators to test bytes for a particular bit pattern. For instance, the AND operator may be used to "mask" all but one of the bits of a status byte at a machine I/O port. The OR operator may be used to "merge" two bytes to create a particular binary value. The following examples will help demonstrate how the logical operators work.

Decimal	Binary	
4 OR 2=6 10 OR 10=10 -1 OR -2=-1 1111111	111111 AND 001111 AND 111111111 AND 000100 AND 001010 OR 111111111 OR	010000=010000 001110=001110 001000=000100 000010=000110 001010=001010 11111111
The bit complement of sixteen zeros is sixteen ones, which is the two's complement representation of -1.		
NOT X=-(X+1)		nplement of any bit complement

FUNCTIONAL OPERATORS

When a function is used in an expression, it calls a predetermined operation that is to be performed on an operand. GWBASIC has "intrinsic" functions that reside in the system, such as SQR (square root) or SIN (sine). All GWBASIC intrinsic functions are described in the Reference Section.

GWBASIC also allows "user-defined" functions that are written by the programmer. (See "DEF FN" Statement in the Reference Section.)

STRING OPERATORS

Strings may be concatenated by using +.

Example

10 A\$="FILE": B\$="NAME"
20 PRINT A\$+B\$
30 PRINT "NEW"+A\$+B\$
RUN
FILENAME
NEW FILENAME
Ok

Strings may be compared using the same relational operators that are used with numbers:

String comparisons are made by taking one character at a time from each string and comparing the ASCII codes. If all the ASCII codes are the same, the strings are equal. If the ASCII codes differ, the lower code number precedes the higher. If during string comparison the end of one string is reached, the shorter string is said to be smaller. Leading and trailing blanks are significant.

Example

```
"AA"<"AB"
"FILENAME" = "FILENAME"
"X&">"X#"
"CL">"CL"
"kg">"KG"
"SMYTH"<"SMYTHE"
B$<"9/12/78" where B$ = "8/12/78"
```

Thus, string comparisons can be used to test string values or to alphabetize strings. All string constants used in comparison expressions must be enclosed in quotation marks.

Note that lower case letters have higher ASCII codes than upper case letters.

and the control of the second of the control of the

+ - 4e

4

Disk File Handling

- Device Independent Input/Output
- How MS-DOS Keeps Track of Your Files
- File Specification
- Commands for Program Files
- Disk Data Files Sequential and Random Access

DEVICE INDEPENDENT INPUT/OUTPUT

GWBASIC provides device-independent input/output that permits flexible approaches to data processing. Using device independent I/O means that the syntax for access is the same for any device.

The following statements, commands, and functions support device-independent I/O (see individual descriptions in the Reference section):

BLOAD BSAVE CHAIN CLOSE EOF GET INPUT INPUT\$ LINE INPUT LIST LOAD LOC LOF
MERGE
OPEN
POS
PRINT
PRINT USING
PUT
RUN
SAVE
WIDTH
WRITE

HOW MS-DOS KEEPS TRACK OF YOUR FILES

A file is a collection of information. The names of files are kept in directories on the fixed disk or the diskette. These directories also contain information on the size of the files, their location on the fixed disk or the diskette, and the dates that they were created and updated. The directory you are working in is called your current directory.

An additional system area is called the File Allocation Table. It keeps track of the space used by your files. It also allocates the free space on your fixed disk or diskette so that you can create new files.

These two system areas, the directories and the File Allocation Table, enable MS-DOS to recognize and organize the files on your disks. The File Allocation Table is created on a new fixed disk or diskette when you format it with the MS-DOS FORMAT command, and one empty directory is created, known as the root directory.

To use the information in a file you must OPEN the file to tell GWBASIC where the information is. You may then use the file for input and/or output.

Using GWBASIC, any type of input/output may be treated like I/O to a file, whether you are actually using a disk or diskette file, or are communicating with another computer or a peripheral device. Thus some I/O statements, functions and commands allow you to specify or refer to either a file or a device (e.g. OPEN, LIST, CLOSE, etc...).

FILE SPECIFICATION

FILE NUMBERS

Up to 15 files, numbered 1 to 15, can be opened by GWBASIC at one time. The number of files that can be opened is specified using the /F: option on the GWBASIC command. A file number is associated with a file when the file is opened.

NAMING FILES

Each file is uniquely identified. The filename is a string expression with the following format:

"[device:|filename"

The device name (or "device") tells GWBASIC on which device to look for the file, and the filename tells GWBASIC which file to look for on that device. The device name is optional. If omitted the default device is assumed. Note the colon (:), indicated above, must be used whenever a device is specified.

A file name can comprise:

- one to eight characters (for legal characters see below). For example NEWFILE.
- one to eight characters, followed by a period (.) and a one to three character file name extension. For example NEWFILE.EXE.

A file name may be made up of any of the following characters:

A-Z	0-9	\$	2	#	~
%	£	[)	-	
0	^	{	}	ļ	

Alphabetic characters within the file name can be entered in upper or lower case, but MS-DOS will translate lower case letters into upper case.

GWBASIC supplies the extension .BAS if no extension is given, but NAME and KILL do not follow this rule: they do not supply any extension.

File specification for communications devices is slightly different. The filename is replaced with a list of options specifying such things as line speed. Refer to OPEN COM statement in the Reference section for details.

Remember that if you use a string constant for the filename, you must enclose it in quotation marks. The only exception to this rule is the MS-DOS GWBASIC command, where a filename is a string constant not included in quotation marks.

For example in GWBASIC, you would type:

RUN "B:ARSENAL.RED"

but from MS-DOS you use:

A>gwbasic b:arsenal.red

NAMING DEVICES

The device name consists of up to four characters and is always followed by a colon (:). The colon must always be used whenever a device is specified. The device name may be one of the following:

A: first diskette drive (Input and Output)

B: second diskette drive (Input and

Output)

C: first hard disk drive (Input and

Output)

D: second hard disk drive (Input and

Output)

KYBD: key board (Input only) **SCRN:** screen (Output only) **LPT1:** first printer (Output only)

LPT1: first printer (Output only)
second printer (Output only)
third printer (Output only)
COM1: RS232 Communications 1

(Input and Output)

COM2: RS232 Communications 2

(Input and Output)

COM3: RS232 Communications 3

(Input and Output)

COM4: RS232 Communications 4

(Input and Output)

DIRECTORY PATHS

With GWBASIC the user can organize a disk in such a manner that files that are not part of his current task do not interfere with that task.

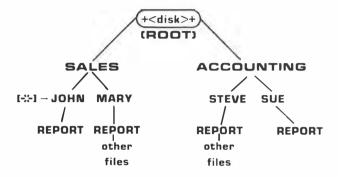
Previously, only a single directory was supported that contained all files on a disk. MS-DOS extends this concept to allow a directory to contain both files and directories and to introduce the notion of the "current" directory.

To specify a file, the user could use one of two methods: either specify a path from the root directory to the file, or specify a path from the current directory to the file. A "Directory Path" (or "pathname") is a series of directory names separated by "and ending with a file name. A pathname that starts at the root begins with the ".

There are two special directory entries in each directory, denoted by '.' and '..'. They specify the directory itself ('.') or the parent of the directory ('..'). The root directory's parent is itself.

Let us take a hypothetical example.

In a particular business, both sales and accounting share a computer with a large disk and the individual employees use it for preparation of reports and maintaining accounting information. One would naturally view the organization of files on the disk as shown on the next page.



Using a directory structure like the hierarchy above, and assuming that the current directory is at point [*] (directory JOHN), to reference the REPORT under JOHN, the following are equivalent:

REPORT

\SALES\JOHN\REPORT

To refer to the REPORT under MARY, supposing that JOHN is still the current directory, the following are equivalent:

../MARY/REPORT /SALES/MARY/REPORT

To refer to the REPORT under SUE, supposing that JOHN is still the current directory, the following are equivalent:

../../ACCOUNTING/SUE/REPORT /ACCOUNTING/SUE/REPORT

CURRENT DIRECTORY

GWBASIC remembers a default directory (called the "current" directory) for each drive on your system. This is the directory that GWBASIC will search if you enter a filename without specifying which directory the file is in. A single directory is created on a disk when it is formatted. That directory is called the "root" directory. You can create other directories by entering the MKDIR command, or remove directories by entering the RMDIR command (see the Reference section.) The CHDIR command allows you to change the current directory. Just after formatting a diskette the ROOT directory is the current directory.

If a "pathname" begins with a backslash (\), GWBASIC starts its search from the "root"; otherwise it starts its search from the current directory. The "pathname" you specify can be a sequence of directory names starting either with the "root," or with the current directory. If the file belongs to the current directory you only need to specify the file.

There is no restriction on the depth of a tree (the length of the longest path from root to leaf) except in the number of allocation units available. The root directory will have a fixed maximum number of entries, 64 or 112 files for a diskette. The maximum number of files in a hard disk root directory depends on the size of the MS-DOS partition on the disk.

Other sub-directories can also be accessed via the root directory, and these in turn can branch off to further files and sub-directories. The only limit is the amount of available space on the disk.

Old (pre 2.0) disks will appear to MS-DOS 2.0 as having only a root directory with files in it and no sub-directories whatever.

Each directory can also contain file and directory names that also appear in other directories.

Pathnames can be used for the following commands:

BLOAD	GWBASIC(*)	NAME
BSAVE	KILL	OPEN
CHAIN	LOAD	RMDIR
CHDIR	MERGE	RUN
FILES	MKDIR	SAVE

(*) Used to initialize GWBASIC. This is an MS-DOS command (not a GWBASIC command).

A "pathname" may be considered as an extension of "filename" and is a string expression of the form:

[device:][\directory][\directory]...[\] filename

or

 $[device:][\directory][\directory]...[\directory]$

All characters that are valid for a filename are also valid for a directory name.

Examples (supposing **JOHN** is the current directory):

B:\SALES\MARY\REPORT

B:..\MARY\REPORT

The GWBASIC command and some GWBASIC commands allow you to specify a file by either a "filename" or a "pathname" LOAD, MERGE, NAME, OPEN, RUN and SAVE.

Some GWBASIC commands allow you to use only the latter form of a "pathname." They are: MKDIR, RMDIR, and CHDIR.

The FILES command allows you to use both forms to display either all files residing on a directory or a single file, or a group of files by using wild cards (* and/or ?).

A "pathname" may not contain more than 63 characters. Pathnames longer than 63 characters will give a "Bad Filename" error.

Specifying a "pathname" where only a "filename" is legal, or placing a "device" other than at the beginning of the "pathname" will result in a "Bad Filename" error.

If you use a string constant for the "pathname," you must enclose it in quotation marks. Only the GWBASIC command specifies pathnames as literal strings not included in quotation marks.

COMMANDS FOR PROGRAM FILES

The following list reviews the commands and statements used in program file manipulation.

With GWBASIC the asterisk (*) and question mark (?) can be used as wild cards with the FILES and KILL commands.

SAVE filename [, $\{A|P\}$] or SAVE pathname [, $\{A|P\}$]

Writes to disk the program that currently resides in memory. Option A writes the program as a series of ASCII characters (otherwise, GWBASIC or uses a compressed binary format); option P writes the program in a protected form. (See Protected Files in this chapter.)

LOAD filename [,R] or LOAD pathname [,R]

Loads the program from disk into memory. Option R runs the program immediately. LOAD always deletes the current contents of memory and closes all files before loading. If R is included, however, open data files are kept open. Thus, programs may be chained or loaded in sections and access the same data files. LOAD filename, R and RUN filename, R are equivalent.

RUN filename [,R]

or

RUN pathname [,R]

Loads the program from disk into memory and runs it. RUN deletes the current contents of memory and closes all files before loading the program. If the R option is included, however, all open data files are kept open. RUN filename, R and LOAD filename, R are equivalent.

MERGE filename

or

MERGE pathname

Loads the program from disk into memory but does not delete the current contents of memory. The program line numbers on disk merge with the line numbers in memory. If two lines have the same number, only the line from the disk program is saved. After a MERGE command, the merged program resides in memory, and GWBASIC returns to command level.

KILL filename

or

KILL pathname

Deletes the file from the disk. The filename may be a program file, or a sequential or random access data file.

NAME{filename} AS filename

or

NAME {pathname} AS {filename}

Changes the name of a disk file. NAME may be used with any disk file.

PROTECTED FILES

If you want to save a program in an encoded binary format, use the Protect option with the SAVE command. For example:

SAVE "MYPROG",P

Because a program saved in this manner cannot be listed or edited, you may want to save an unprotected copy of the program for these purposes.

DISK DATA FILES — SEQUENTIAL AND RANDOM ACCESS

Two types of disk data files can be created and accessed by a GWBASIC program:

- sequential files
- random access files

SEQUENTIAL FILES

Sequential files are easier to create than random access files but limit flexibility and speed when accessing the data. The data written to a sequential file is in the form of ASCII characters which are loaded and stored, one item after another (sequentially), in the order they are sent.

The statements and functions used with sequential files are as follows:

CLOSE
EOF
INPUT\$
INPUT#
LINE INPUT#
LOC
LOF
OPEN
PRINT#
PRINT# USING
WRITE#

See the Reference section of this manual for more information on these statements and functions.

CREATING A SEQUENTIAL FILE

The following program steps are required to create a sequential file and access the data in the file:

• OPEN the file in "O" (Output) mode.

OPEN "O",#1,"DATA"

 Write data to the file using the WRITE# statement. (You may use the PRINT# statement instead; refer to the PRINT# statement in the Reference section.)

WRITE#1,A\$,B\$,C\$

• If you have opened a file in the "O" mode, to access the data in the file, you must CLOSE the file and reOPEN it in "I" (Input) mode.

CLOSE #1 OPEN "I",#1,"DATA"

 Use the INPUT# statement to read data from the sequential file to the program.

INPUT#1,X\$,Y\$,Z\$

A program that creates a sequential file can also write formatted data to the disk with the PRINT# USING statement. For example, the statement

PRINT#1,USING"####.##,";A,B,C,D

could be used to write numeric data to disk without explicit delimiters. The comma (,) at the end of the format string serves to separate each item in the disk file.

The LOC function, when used with a sequential file, returns the number of sectors that have been written to or read from the file since it was opened. For example,

100 IF LOC(1)>50 THEN STOP

would end program execution if more than 50 sectors had been written to, or read from, file #1 since it was opened.

Program 1 is a short program that creates a sequential file, named "DATA," from information you input at the keyboard.

10 OPEN "O",#1,"DATA"
20 INPUT "NAME";N\$
25 IF N\$ = "DONE" THEN END
30 INPUT "DEPARTMENT";D\$
40 INPUT "DATE HIRED";H\$
50 PRINT#1,N\$;",";D\$;",";H\$
60 PRINT:GOTO 20
RUN
NAME? MICKEY MOUSE
DEPARTMENT? AUDIO/VISUAL AIDS
DATE HIRED? 01/12/72

NAME? SHERLOCK HOLMES DEPARTMENT? DATE HIRED? 12/03/65

NAME? EBENEEZER SCROOGE DEPARTMENT? DATE HIRED? 04/27/78

NAME? SUPER MAN DEPARTMENT? DATE HIRED? 08/16/78

NAME? DONE Ok

ACCESSING A SEQUENTIAL FILE

Program 2 accesses the file "DATA" that was created in Program 1 and displays the name of everyone hired in 1978.

10 OPEN "I",#1,"DATA"
20 INPUT#1,N\$,D\$,H\$
30 IF RIGHT\$(H\$,2)="78" THEN PRINT N\$
40 GOTO 20
RUN
EBENEEZER SCROOGE
SUPER MAN
Input past end in 20
Ok

The program reads, sequentially, every item in the file. When all the data has been read, line 20 causes an "Input past end" error. This error can be avoided, however, by inserting an additional line (line 15 shown below) which uses the EOF function to test for end-of-file.

15 IF EOF(1) THEN END

Then change line 40 to GOTO 15.

ADDING DATA TO A SEQUENTIAL FILE

As soon as a sequential file is opened on disk in "O" mode, its current contents are destroyed. In order to add more data to the file it is necessary to use the OPEN statement with the APPEND mode, as described in the Reference section of this manual.

RANDOM ACCESS FILES

Creating and accessing random access files requires more program steps than with sequential files, but there are advantages to using random access files. One advantage is that random access files require less room on the disk, because GWBASIC stores them in a packed binary format. (A sequential file is stored as a series of ASCII characters.)

The biggest advantage to random access files is that data can be accessed at random, i.e., anywhere on the disk. It is not necessary to read through all the information on disk with random access files, as with sequential files. This is possible because the information is stored and accessed in distinct units called records and each record is numbered.

The statements and functions that are used with random access files are:

CLOSE	LOF
CVD	MKD\$
CVI	MKI\$
CVS	MKS\$
FIELD	OPEN
GET	PUT
LSET	RSET
LOC	

CREATING A RANDOM ACCESS FILE

Creation of a random access file requires the following program steps.

1 OPEN the file for random access ("R" mode). Always use the "R" (Random) mode for random access files. "R" allows you to perform both input and output operations on a file.

This example specifies a record length of 32 bytes. If the record length is omitted, the default is 128 bytes.

OPEN "R",#1,"FILE",32

2 Use the FIELD statement to allocate space in the random buffer for the variables that will be written to the random file.

FIELD #1,20 AS N\$, 4 AS A\$,8 AS P\$

3 Use the LSET command to move the data into the random buffer. Numeric values must be made into strings when placed in the buffer. To do this, use the "make" functions. MKI\$ makes an integer value into a string, MKS\$ does the same for a single precision value, and MKD\$ converts a double precision value. See the Reference section for more information on these functions.

LSET N\$ = X\$ LSET A\$ = MKS\$(AMT) P\$ = TEL\$

Write the data from the buffer to the disk using the PUT statement.

PUT #1,CODE%

The LOC function, with random access files, returns the "current record number." The current record number is one, plus the last record number that was used in a GET or PUT statement. For example, the statement

IF LOC(1)>50 THEN END

ends program execution if the current record number in file#1 is higher than 50.

The following example writes information that is input at the terminal to a random access file.

10 OPEN "R",#1,"FILE",32
20 FIELD #1,20 AS N\$, 4 AS A\$, 8 AS P\$
30 INPUT "2-DIGIT CODE";CODE%
40 INPUT "NAME";X\$
50 INPUT "AMOUNT";AMT
60 INPUT "PHONE";TEL\$:PRINT
70 LSET N\$ = X\$
80 LSET A\$ = MKS\$(AMT)
90 LSET P\$ = TEL\$
100 PUT #1,CODE%
110 GOTO 30

Each time the PUT statement is executed, a record is written to the file. The two-digit code that is input in line 30 becomes the record number.

Note: Do not use a FIELDed string variable in an INPUT or LET statement. This causes the pointer for that variable to point into string space instead of into the random access file buffer.

ACCESSING A RANDOM ACCESS FILE

Reading a random access file requires the following steps.

1 OPEN the file in "R" mode.

OPEN "R",#1,"FILE",32

2 Use the FIELD statement to allocate space in the random buffer for the variables that will be read from the file.

FIELD #1,20 AS N\$, 4 AS A\$,8 AS P\$

3 Use the GET statement to move the desired record into the random buffer.

GET #1,CODE%

4 The data in the buffer may now be accessed by the program. Numeric values must be converted back to numbers using the "convert" functions. CVI converts numeric values to integer values, CVS converts numeric values to single precision values, and CVD converts numeric values to double precision values.

PRINT NS

PRINT CVS(A\$)

The following program accesses the "FILE" that was created in the previous example. When the two-digit code is entered at the terminal, the information associated with that code is read from the file and displayed.

10 OPEN "R",#1,"FILE",32
20 FIELD #1, 20 AS N\$, 4 AS A\$, 8 AS P\$
30 INPUT "2-DIGIT CODE";CODE%
40 GET #1, CODE%
50 PRINT N\$
60 PRINT USING "\$\$###.##";CVS(A\$)
70 PRINT P\$:PRINT
80 GOTO 30

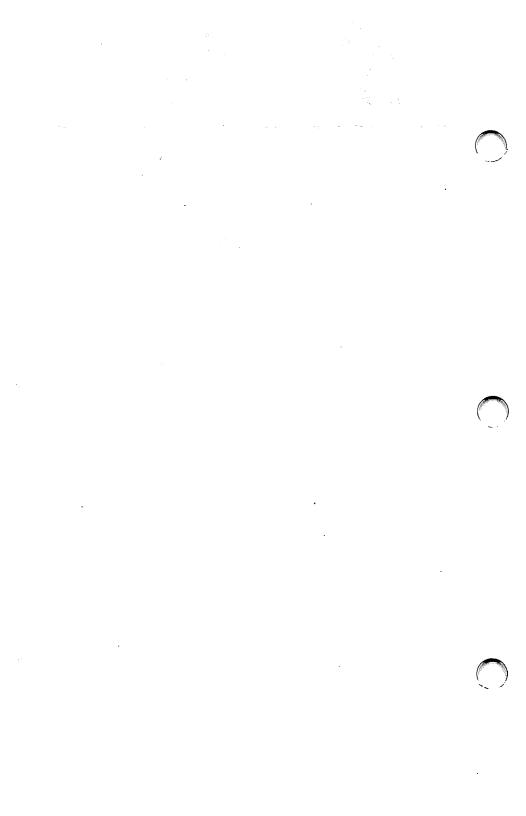
The following example is an inventory program that illustrates random file access. In this program, the record number is used as the part number, and it is assumed the inventory will contain no more than 100 different part numbers. Lines 900 through 960 initialize the data file by writing CHR\$(255) as the first character of each record. This is used later (line 270 and line 500) to determine whether an entry already exists for that part number.

Lines 140 through 210 display the different inventory functions that the program performs. When you type in the desired function number, line 230 branches to the appropriate subroutine.

120 OPEN"R",#1,"INVEN.DAT",39 130 FIELD#1,1 AS F\$,30 AS D\$,2 AS Q\$,2 AS R\$,4 AS P\$ 140 PRINT:PRINT "FUNCTIONS:":PRINT 150 PRINT 1,"INITIALIZE FILE" 160 PRINT 2."CREATE A NEW ENTRY" 170 PRINT 3,"DISPLAY INVENTORY FOR ONE PART" 180 PRINT 4,"ADD TO STOCK" 190 PRINT 5,"SUBTRACT FROM STOCK" 200 PRINT 6,"DISPLAY ALL ITEMS BELOW REORDER LEVEL" 210 PRINT:PRINT:INPUT"FUNCTION";FUNCTION 220 IF (FUNCTION > 1) OR(FUNCTION > 6) THEN PRINT "BAD **FUNCTION NUMBER": GOTO 140** 230 ON FUNCTION GOSUB 900,250,390,480,560,680 240 GOTO 140 **250 REM BUILD NEW ENTRY 260 GOSUB 840** 270 IF ASC(F\$)< >255 THEN INPUT"OVERWRITE":AS: IF AS "Y" THEN RETURN

```
280 LSET FS = CHRS(0)
290 INPUT "DESCRIPTION":DESCS
300 LSET DS = DESCS
310 INPUT "QUANTITY IN STOCK";Q%
320 LSET Q$ = MKI$(Q%)
330 INPUT "REORDER LEVEL"; R%
340 LSET R$ = MKIS(R%)
350 INPUT "UNIT PRICE":P
360 LSET P$ = MKS$(P)
370 PUT #1,PART%
380 RETURN
390 REM DISPLAY ENTRY
400 GOSUB 840
410 IF ASC(F$) = 255 THEN PRINT "NULL ENTRY": RETURN
420 PRINT USING "PART NUMBER ###"; PART%
430 PRINT DS
440 PRINT USING "QUANTITY ON HAND #####":CVI(Q$)
450 PRINT USING "REORDER LEVEL ####" CVI(RS)
460 PRINT USING "UNIT PRICE $$##.##":CVS(P$)
470 RETURN
480 REM ADD TO STOCK
490 GOSUB 840
500 IF ASC(F$) = 255 THEN PRINT "NULL ENTRY": RETURN
510 PRINT DS:INPUT "QUANTITY TO ADD" ;A%
520 Q% = CVI(Q$) + A%
530 LSET Q$ = MKI$[Q%]
540 PUT#1,PART%
550 RETURN
560 REM REMOVE FROM STOCK
570 GOSUB 840
580 IF ASC(F$) = 255 THEN PRINT "NULL ENTRY": RETURN
590 PRINT DS
600 INPUT "QUANTITY TO SUBTRACT"; 5%
610 Q% = CVI(Q$)
620 IF (Q%-S%)<0 THEN PRINT "ONLY";Q%;"IN
STOCK":GOTO 600
```

```
630 Q% = Q%-S%
640 IF Q% = < CVI(R$) THEN PRINT "QUANTITY NOW";Q%;
      " REORDER LEVEL"; CVI(R$)
650 LSET QS = MKIS(Q%)
660 PUT #1.PART%
670 RETURN
680 REM DISPLAY ITEMS BELOW REORDER LEVEL
690 FOR I=1 TO 100
710 GET #1.I
715 IF ASC (F$) = 255 THEN 730
720 IF CVI(Q$) < CVI(R$) THEN PRINT D$; "QUANTITY";
      CVI(Q$) TAB(50) "REORDER LEVEL";CVI(R$)
730 NEXT I
740 RETURN
840 INPUT "PART NUMBER";PART%
850 IF(PART%<1)OR(PART%>100) THEN PRINT "BAD
      PART NUMBER": GOTO 840 ELSE
      GET #1.PART%:RETURN
890 END
900 REM INITIALIZE FILE
910 INPUT "ARE YOU SURE": BS:IF BS "Y" THEN RETURN
920 LSET F$ = CHR$(255)
930 FOR I=1 TO 100
940 PUT #1.I
950 NEXT I
960 RETURN
```



5

Graphics

- Selecting the Screen Attributes
- Text Mode
- Graphics Mode

SELECTING THE SCREEN ATTRIBUTES

The SCREEN statement allows you to switch between text and graphics modes and the WIDTH statement allows you to set the number of columns.

There are three different graphics modes you can select with the SCREEN statement:

- Medium Resolution Mode
- High Resolution Mode
- Super Resolution Mode

They differ only in the number and size of the points displayed and in the number of colors allowed.

The SCREEN statement also allows you (through the "burst" parameter) to enable color in Text or Medium-Resolution Mode (using a color TV set or RGB monitor), and to select the active and display pages in Text Mode (through the "apage" and "vpage" parameters). For a standard monitor, the "burst" parameter has no meaning.

The SCREEN statement must precede any I/O statements to the screen, as it selects the "screen attributes" to be used by subsequent statements. The system assumes SCREEN 0,0,0,0 by default if no screen attributes are specified. This selects 80 columns Text Mode, B/W, and only one display page.

You can also use more than one SCREEN statement to define different screen attributes for different sections of your program.

TEXT MODE

In Text Mode you can display text, i.e., letters, numbers, and all special characters of the GWBASIC character set. You can set the character foreground and background colors using the COLOR (Text) statement. This statement also allows you to create blinking, reverse image, invisible, highlighted, and underscore characters.

Characters are displayed in 25 horizontal lines numbered 1 through 25, from top to bottom. Each line has 40 (or 80) character positions. The WIDTH command allows you to select the number of columns.

The LOCATE statement positions the cursor on the active screen. The cursor column and line coordinates are returned by the POS(0) and CSRLIN functions.

Characters are normally displayed, using the PRINT or PRINT USING statements, at the cursor position from left to right on each line, from line 1 to 24. When the cursor passes to line 25, lines 1 through 24 are scrolled up one line.

Line 25 is usually reserved as a "soft key" display (see KEY statement in the Reference section).

Multiple Display Page

A special feature of Text Mode is multiple display pages. Your system has a 16K-byte screen buffer, of which only 2K is required for Text Mode (or 4K for 80 column width). The buffer is therefore divided into different pages, which can be written to and/or displayed individually. There are 8 display pages in 40 column width and 4 display pages in 80 column width.

Statements, Commands and Functions

The statements, commands and functions available in Text Mode to display text are:

Statements/ Commands	Functions
CLS COLOR (Text) LOCATE (Text) PRINT PRINT USING SCREEN WIDTH WRITE	CSRLIN POS SCREEN SPC TAB

In Text Mode you can use 16 different colors (if color hardware is installed):

0 Black 8 Gray 9 Light Blue 1 Blue 2 Green 10 Light Green 3 Cyan 11 Light Cyan 4 Red 12 Light Red 13 Light Magenta 5 Magenta 14 Yellow 6 Brown

7 White 15 High-intensity White

In a monochrome system only two colors are available (black and white), but you can underline characters, make characters blink, or display high-intensity characters.

GRAPHICS MODE

In Graphics Mode you can draw complex pictures as well as display text. To display text in Graphics Mode you can use all the statements, commands and functions available in Text Mode, with the exception of the COLOR (Text) and LOCATE (Text) statements. In Graphics Mode you have to use the COLOR (Graphics) and LOCATE (Graphics) statements instead. Note also that the CLS and WIDTH statements have different features in Graphics Mode.

In Graphics Mode all points of the screen are addressable in medium, high or super resolution. A point on the screen is called a 'pixel' (a contraction of "picture element"), and a line of pixels is called a "scanline."

To print pictures that you have generated in the Graphics Mode, you must run the MS-DOS GRAPHICS command before running the BASIC program that draws the pictures. See the *User Guide to MS-DOS* for details.

Statements, Commands and Functions

The statements, functions and commands you can use to display pictures are:

Statement/Command

Function

CIRCLE
COLOR (Medium-Resolution Mode)
COLOR (High-Resolution Mode)
COLOR (Super-Resolution Mode)
DRAW
GET (Graphics)
LOCATE (Graphics)
PAINT
PRESET
PSET
PUT (Graphics)
SCREEN
VIEW
WINDOW

PMAP POINT

MEDIUM RESOLUTION MODE

In this mode, there are 320 pixels on the horizontal axis and 200 pixels on the vertical axis. These are numbered from left to right and from top to bottom; thus the upper left corner pixel is (0,0) and the lower right corner pixel is (319, 199).

You can display four colors at a time if a color monitor is used, otherwise the four colors will appear as shades of grey.

Drawing Pictures

When you draw pictures on the screen using the graphics statements (PSET, PRESET, LINE, CIRCLE, PAINT or DRAW), you can specify a color number of 0, 1, 2, or 3. This selects the color from the current "palette" as defined by the COLOR statement.

If you do not specify a color number, the default is the graphics foreground specified by the COLOR statement, or 3 (if no graphics foreground is given).

The COLOR (Medium-Resolution) statement allows you to specify both the color for color number 0, and the "palette" for the three remaining color numbers (1, 2, and 3).

Palette	Color 1	Color 2	Color 3
0	Green	Red	Yellow
1	Cyan	Magenta	White

If color is disabled the use of memory is identical: the modes differ only in that the two bits of a pixel are interpreted differently by the hardware: medium resolution B/W displays 4 shades of grey.

Displaying Characters

When you display characters in Medium Resolution Mode, the size of the characters is the same as in Text Mode when you specify a 40-column width. The character foreground color is set by the "tforeground" parameter in the COLOR statement (that defaults to color number 3). The character background is set by the "background" parameter in the COLOR statement (that defaults to color number 0, i.e., Black).

If color is disabled the character foreground will be 1 (White) and the character background 0 (Black).

HIGH RESOLUTION MODE

In this mode, there are 640 pixels on the horizontal axis and 200 pixels on the vertical axis. These are numbered from left to right and top to bottom; thus the upper left corner pixel is (0,0) and the lower right corner pixel is (639, 199).

There are only two colors: black (color number 0) and white (color number 1).

Drawing Pictures

When you draw pictures using the graphics statements, you can still specify a color number 0, 1, 2, or 3.

A color of 0 indicates black and a color of 1 white. A color of 2 is treated as 0, and 3 is treated as 1.

If you do not specify a color number, the default is the graphics foreground specified by the COLOR statement, or 1 (if no graphics foreground is given).

The COLOR statement allows you to specify the graphics foreground color.

Displaying Characters

The size of the characters is the same as in 80-column Text Mode.

The character foreground color is 1 (white) and the background color is 0 (black).

SUPER RESOLUTION MODE

In this mode, there are 640 pixels on the horizontal axis and 400 pixels on the vertical axis. These are numbered from left to right and top to bottom; thus the upper left corner pixel is (0,0) and the lower right corner pixel is (639, 399).

There are only two colors: black (color number 0) and white (color number 1).

Drawing Pictures

When you draw pictures using the graphics statements, you can still specify a color number of 0, 1, 2, or 3.

A color number of 0 indicates black and a color number of 1 indicates white. A color number of 2 is treated as 0, and a color number of 3 is treated as 1.

If you do not specify a color number, the default is the graphics foreground specified by the COLOR statement, or 1 (if no graphics foreground is given).

The COLOR (Super Resolution) statement allows you to specify the graphics foreground color. The COLOR statement also allows you to specify 'inverse video', when you display characters.

Displaying Characters

The size of the characters is the same as in 80-column Text Mode.

The character foreground color is 1 (white) and the character background 0 (black), unless you specify 'inverse video' by the COLOR statement.

SCREEN COORDINATES

Graphics images are positioned on the screen in accordance with screen coordinates. These coordinates comprise two parameters generally referred to as x and y, where x defines the horizontal screen position and y defines the vertical screen position. Screen coordinates may be of two types:

- absolute coordinates
- relative coordinates

Whereas absolute coordinates refer to the actual position of a pixel on the screen, relative coordinates indicate the position of a pixel relative to the coordinates of the last pixel referenced. The x and y relative coordinates are therefore 'offset' values from the last pixel referenced (known as the 'current point').

Screen coordinates are described fully in the Reference section (refer particularly to the WINDOW statement); however, the following example illustrates the use of both types of coordinates:

10 SCREEN 1 20 PSET (100,50) 'absolute coordinates 30 PSET STEP (10,-5) 'relative coordinates

This program example illuminates two pixels on the screen: the first at coordinates (100,50) and the second at actual coordinates (110,45.)

VIEW STATEMENT

The VIEW statement allows the definition of subsets of the viewing surface. These are called "viewports." Onto these the contents of a window are mapped. Initially RUN or VIEW, with no arguments, define the whole screen as a viewport. Refer to the Reference section for a full description of VIEW.

WINDOW STATEMENT

WINDOW allows you to draw lines, graphs, or objects in space not bounded by the physical limits of the screen. This is done by using programmer-defined coordinates called "World coordinates."

A world coordinate is any valid single precision floating point number pair. GWBASIC then converts world coordinate pairs into the appropriate physical coordinate pairs for subsequent display within screen space. To make this transformation from world space to the physical space of the viewing surface (screen), GWBASIC must know what portion of the unbounded (floating point) world coordinate space contains the information you want to be displayed.

This rectangular region in world coordinate space is called a WINDOW.

Refer to the Reference section for a full description of the WINDOW statement.

DISPLAYING POINTS

The most elementary graphic function is that of illuminating the position of a single point (or 'pixel') in a specified color. This is achieved using the PSET and PRESET statements. The POINT function allows you to know the color number of a specified pixel. Refer to a full description of these in the Reference section.

DRAWING AND COLORING LINES, RECTANGLES, OBJECTS, CIRCLES, ARCS, ELLIPSES

The LINE statement permits the drawing of lines or rectangles. The DRAW statement, governed by "movement commands" such as up, down, left, and right, lets you draw any object. Circles, arcs and ellipses can be drawn using the CIRCLE statement, and the PAINT statement allows any object to be filled with color(s).

Refer to statements: LINE, CIRCLE, GET, PUT (graphics), PAINT, and DRAW in the Reference section for a complete description.

LINE CLIPPING

The graphics statements CIRCLE, LINE, PAINT, POINT, PSET, PRESET, and WINDOW use "line clipping." This simply means that lines which cross the screen or viewport are "clipped" at the boundaries of the viewing area. Only the points plotted within the screen or viewport are visible.

6 Asynchronous Communications

- Opening Communications Files
- Communication I/O
- Communication I/O Functions

OPENING COMMUNICATIONS FILES

The OPEN COMmunications statement allocates a buffer for input and output in a similar manner as the OPEN statement for disk files. Refer to the **OPEN COM** Statement in the Reference section for a full description.

COMMUNICATION I/O

Since the communication port is opened as a file, all Input/Output statements that are valid for disk files are valid for COM.

COM sequential input statements are the same as those for disk files. They are: INPUT #, LINE INPUT #, and the INPUT\$ function.

COM sequential output statements are the same as those for disk, and are: PRINT #, PRINT # USING, and WRITE #.

Refer to the descriptions of these statements in the Reference section for details of coding syntax and usage.

The GET and PUT statements are only slightly different for COM files. See the GET (COM Files) and PUT (COM Files) statements described in the Reference section.

COMMUNICATION I/O FUNCTIONS

The most difficult aspect of asynchronous communication is being able to process characters as fast as they are received. At rates above 2400 bps, it may be necessary to suspend character transmission from the host computer long enough to catch up. This can be done by sending XOFF (CHR\$(19)) to the host and XON (CHR\$(17)) when ready to resume.

GWBASIC provides three functions which help in determining when an over-run condition is imminent. These are:

LOC(f)

Returns the number of characters in the input buffer waiting to be read. The input buffer can hold more than 255 characters (determined by the /C: switch). If there are more than 255 characters in the buffer, LOC(f) returns 255. Since a string is limited to 255 characters, this practical limit means that you do not have to test for string size before reading data into it. If fewer than 255 characters remain in the buffer, LOC(f) returns the actual count.

LOF(f)

Returns the amount of free space in the input buffer. That is, size-LOC(f), where 'size' is the size of the communications buffer as set by the /C: option. LOF may be used to detect when the input buffer is reaching its maximum capacity.

EOF(f)

If true (-1), indicates that the input buffer is empty. Returns false (0) if any characters are waiting to be read.

Possible Errors

- Communication Buffer Overflow If a read is attempted after the input buffer is full, (i.e. LOF(f) returns 0).
- Device I/O Error
 If any of the following line conditions are
 detected on reception: Overrun Error (OE),
 Framing Error (FE), or Break Interrupt (BI).
 The error is reset by subsequent inputs but the
 character causing the error is lost.
- Device Fault
 If Data Set Ready (DSR) is lost during I/O.

THE INPUTS FUNCTION FOR COM FILES

The INPUT\$ function is preferable to the INPUT# and LINE INPUT# statements when reading COM files, since all ASCII characters may be significant in communications. INPUT# is least desirable because input stops when a comma (,) or CR is received and LINE INPUT# terminates when a CR is received.

INPUT\$ allows all characters read to be assigned to a string. INPUT\$ (n,f) will return n characters from the #f file. The following statements are therefore the most efficient for reading a COM file:

10 WHILE NOT EOF(1) 20 A\$ = INPUT\$(LOC(1),#1)

30 . . .

40 . . . Process data returned in A\$

50 . .

60 WEND

The above statements return the characters in the buffer into A\$ and process them, provided there are characters in the buffer. If there are more than 255 characters, only 255 will be returned at a time to prevent String Overflow. If this is the case, EOF(1) is false and input continues until the input buffer is empty. The sequence of events is therefore simple, concise, and fast.

AN EXERCISE IN COMMUNICATION I/O

The following program enables your Personal Computer to be used as a conventional terminal. Besides Full Duplex communication with a host, the TTY program allows data to be downloaded to a file. Conversely, a file may be uploaded (transmitted) to another machine.

In addition to demonstrating the elements of Asynchronous Communication, this program should be useful in transferring GWBASIC programs and data to and from your system.

```
10 SCREEN 0.0:WIDTH 80
15 KEY OFF:CLS:CLOSE
20 DEFINT A-Z
25 LOCATE 25.1
30 PRINT STRINGS(60." ")
40 FALSE = 0:TRUE NOT FALSE
50 MENU = 5 'Value of MENU key (ctrl-E)
60 XOFFS = CHRS(19):XONS = CHRS(17)
100 LOCATE 25,1:PRINT "Async TTY Program";
110 LOCATE 1,1:LINE INPUT "Speed? ":SPEEDS
120 COMFILS = "COM1:" + SPEEDS + ".E.7"
130 OPEN COMFILS AS #1
140 OPFN "SCRN:" FOR OUTPUT AS #3
200 PAUSE = FAI SE
210 AS=INKEYS: IF AS=" " THEN 230
220 IF ASC(AS) = MENU THEN 300 ELSE PRINT #1.AS:
230 IF EOF(1) THEN 210
240 IF LOC(1)>128 THEN PAUSE = TRUE: PRINT #1,XOFFS:
250 A$ = INPUT$(LOC(1),#1)
253 LINEFEED = 0
255 LINEFEED = INSTR (LINEFEED + 1. AS.CHRS(10))
257 IF LINEFEED = 0 THEN MIDS(AS, LINEFEED, 1) = CHR$(0):GOTO 255
260 PRINT #3.AS::IF LOC(1)>0 THEN 240
270 IF PAUSE THEN PAUSE = FALSE:PRINT #1.XONS:
280 GOTO 210
300 LOCATE 1.1:PRINT STRINGS(30." "1:LOCATE 1.1
310 LINE INPUT"FILE? ":DSKFILS
400 LOCATE 1,1:PRINT STRING$(30," "):LOCATE 1,1
410 LINE INPUT"(T)RANSMIT OR (R)ECEIVE? ";TXRX$
420 IF TXRX$="T" THEN OPEN DSKFIL$ FOR INPUT AS #2:GOTO
430 OPEN DSKFILS FOR OUTPUT AS #2
440 PRINT #1.CHR$(13):
500 IF EOF(1) THEN GOSUB 600
510 IF LOC(1) > 128 THEN PAUSE = TRUE: PRINT #1,XOFF$;
520 A$ = INPUT$(LOC(1),#1)
530 PRINT #2,A$;:IF LOC(1)>0 THEN 510
540 IF PAUSE THEN PAUSE = FALSE:PRINT #1,XOFF$;
550 GOTO 500
600 FOR I=1 TO 5000
610 IF NOT EOF(1) THEN I = 9999
630 IF I=9999 THEN RETURN
640 CLOSE #2:CLS:LOCATE 25,10:PRINT "* Download complete *";
650 RETURN 200
1000 WHILE NOT EOF(2)
1010 A$ = INPUT$(1,#2)
1020 PRINT #1,A$;
1030 WEND
1040 PRINT #1,CHR$(26); 'CTRL-Z to make close file.
1050 CLOSE #2:CLS:LOCATE 25,10:PRINT "** pload complete **";
1060 GOTO 200
9999 CLOSE:KEY ON
```

Line Comments

- 10 Sets the screen to Black and White Text Mode and sets the Width to 80.
- Turns off the Soft Key Display, clears the screen, and makes sure that all files are closed.

Asynchronous implies character I/O as opposed to line or Block I/O. Therefore, all PRINT's (either to the COM file or screen) are terminated with a semicolon (;). This cancels the CR LF normally issued at the end of a PRINT statement.

- 20 Defines all numeric variables as INTEGER. Primarily for the benefit of the subroutine at 600-620. Any program looking for speed optimization should use integer counters in loops where possible.
- 25-30 Clears the 25th line starting at column 1.
 - 40 Defines Boolean TRUE and FALSE.
 - 50 Defines the ASCII (ASC) value of the MENU key.
 - 60 Defines the ASCII XON, XOFF characters.
- 100-130 Prints program-id and asks for baud rate (speed). Opens Communications to file number 1, Even parity, 7 data bits.

This section can be modified to check for valid baud rates before continuing.

- 200-280 This section performs Full Duplex I/O between the Video Screen and the device connected to the RS232 connector as follows:
 - Read a character from the keyboard into A\$. Note that INKEY\$ returns a null string if no character is waiting.
 - If no character is waiting then check to see if any characters are being received. If a character is waiting at the keyboard then:
 - If the character was the MENU Key, then the user is ready to download a file, so retrieve the file name.
 - If character (A\$) is not the MENU key then send it by writing to the communication file (PRINT #1...).
 - At 230, check if any characters are waiting in COM buffer. If not, then go back and check keyboard.
 - At 240, if more than 128 characters are waiting, then set the PAUSE flag, thereby suspending input and send XOFF to the host, thus stopping further transmission.
 - At 253-257, strip out linefeed characters before sending buffer contents to the screen. Otherwise the PC executes a LF with each CR, resulting in double spacing.

- At 250-260, read and display the contents of COM buffer on screen until empty. Continue to monitor (in 240). Suspend transmission in the event of an interface delay.
- Finally, resume host transmission by sending XON only if suspended by previous XOFF.
 Repeat process until MENU Key struck.
- 300-310 Retrieves the name of the Disk File from which the information is to be downloaded. Opens the file to file number 2.
- 400-420 Asks if file named is to be transmitted (Uploaded) or received (Downloaded).
- 430-440 Sends a **CR** to the host to begin the download. This program assumes that the last command sent to the host to begin such a transfer was missing only the terminating **CR**.

- When no more characters are being received (LOC(x) returns 0), then performs a time-out routine (explained later).
- 510 Again, if more than 128 characters are waiting, this line signals a pause, and in the meantime sends XOFF to the host.
- 520-530 Reads all characters in the COM buffer (LOC(x)) and writes them to disk (PRINT #2..).
- 540-550 If a pause was issued, restart host by sending XON and clear the pause flag. Continue process until no characters are received for a pre-determined time.
- 600-650 This is the time-out subroutine. The FOR loop count was determined by experimentation. In short, if no character is received from the host for 17-20 seconds, then transmission is assumed complete. If any character is received during this time (line 610) then set I well above FOR loop range to exit loop and then return to caller. If host transmission is complete, close the disk file

1000-1060 Transmit routine. Until end of disk file do:

Read one character into A\$ with INPUT\$ statement. Send character to COM device in 1020. Send a CTRL Z at end of file in 1040 if receiving device needs to close its file. Finally, in lines 1050 and 1060, close the disk file, print completion message, and go back to conversation mode in line 200.

9999 Presently not executed. As an exercise, add some lines to the routine 400-420 to optionally exit the program via line 9999. This line closes the COM file which is left open and restores the Soft Key Display.

.

7

Command References

- Introduction
- Commands, Statements, and Functions with Examples

INTRODUCTION

The following GWBASIC commands, statements, and functions are described in this chapter.

ABS Returns the absolute value of a numeric

expression.

ASC Returns the ASCII decimal code for the first

character of a given string.

ATN Returns the arctangent of the argument.

AUTO Generates a line number after every carriage

return. AUTO is used only for entering

programs.

BEEP Activates the bell.

BLOAD Loads a memory image file into memory.

BSAVE Saves sections of the main memory on the

specified file.

CALL Transfers control to a machine language

subroutine. Passes unsegmented addresses.

CALLS Transfers control to a machine language

subroutine. Passes segmented addresses.

CDBL Converts a given numeric expression to a double

precision number.

CHAIN Transfers control and passes variables to another

program.

CHDIR Changes the current directory.

CHR\$	Returns a	one-character	string	whose .	ASCII

decimal code is the value of the argument passed

to this function.

CINT Converts any numeric argument to an integer by

rounding the fractional portion.

CIRCLE Draws a circle or an ellipse with the specified

center and radius. (Graphics Mode.)

CLEAR Clears all numeric variables to zero, all string

variables to null, and closes all open files.

Options set the highest memory location available for use by GWBASIC and set the

amount of stack space.

CLOSE Terminates I/O to a file or device.

CLS Erases all or part of the screen.

COLOR In the Text Mode, sets the foreground and

background colors. In Graphics Mode, defines the

background and foreground palette colors.

COM(n) Enables or disables event trapping of

communications activity on the specified channel.

COMMON Defines the common area that is not erased by a

CHAINed program, and allows you to pass variables from one program to another.

CONT Resumes program execution after a

CTRL-BREAK has been typed or a STOP or

END statement has been executed.

COS

CSNG	Converts any numeric argument to a single precision number.
CSRLIN	Returns the current line (row) position of the cursor.
CVD	Converts an eight-byte string to a double precision number.
CVI	Converts a two-byte string to an integer.
CVS	Converts a four-byte string to a single precision number.
DATA	Creates an "internal file" of data items that can be assigned to program variables using the READ statement.
DATE\$	The DATE\$ statement sets the current date. The DATE\$ function retrieves the current date.
DEF FN	Defines and names user-written functions.

Assigns the current segment of memory.

by specifying the starting address.

letter(s) specified.

Erases program lines.

Enables access to a machine language subroutine

Declares the variable type in accordance with the

Returns the cosine of the argument.

DEF SEG

DEF USR

DEFtype

DELETE

•	
DIM	Specifies the array name, the number of dimensions, and the subscript upper bound for each dimension. May specify one or more arrays.
DRAW	Draws an object as specified by the contents of a string expression. (Graphics Mode.)
EDIT	Lets you change a program line.
END	Terminates program execution, closes all open data files, and returns to the command level.
ENVIRON	Allows a modification of parameters in GWBASIC's Environment String Table.
ENVIRON\$	Retrieves the specified Environment String from GWBASIC's Environment String Table.
EOF	Indicates that the end of file has been reached.
ERASE	Releases space and variable names previously reserved for arrays.
ERDEV	An integer function that contains the error code returned by the last device to declare an error.
ERDEV\$	A string function that contains the name of the device driver that generated the error.
ERL	Returns the number of the line that contains the error.
ERR	Returns an error code.

ERROR	Simulates the occurrence of a GWBASIC error, or generates a user-defined error.
EXP	Returns "e" (base of natural logarithms) to the power of the argument.
FIELD	Allocates space for variables in a random file buffer.
FILES	Displays the names of the files in the specified directory.
FIX	Returns the truncated integer part of the argument.
FORNEXT	Allows a series of statements to be performed in a loop a specified number of times.
FRE	Returns the number of bytes in memory not being used by GWBASIC.
GET(COM)	Reads a specified number of bytes into the communications buffer.
GET(Files)	Reads a record from a random disk file into a random buffer.
GET(Graphics)	Reads graphics images from the screen.
GOSUB RETURN	GOSUB transfers control to a GWBASIC subroutine by branching to the specified line. RETURN transfers control to the statement following the most recent GOSUB (or ONGOSUB) executed, or to a specified line.
GOTO	Transfers control to a specified program line.
GWBASIC	Initializes GWBASIC and the operating environment. (MS-DOS command.)

HEX\$ Returns a string that represents the hexadecimal value of the decimal argument.

IF...GOTO...ELSE IF...THEN...ELSE

> Makes a decision regarding program flow based on the result of a specified condition.

INKEY\$ Returns either a one- or two-character string read from the keyboard.

INP Returns the byte read from a port.

INPUT Allows input from the keyboard during program

execution.

INPUT# Reads data items from a sequential disk file and

assigns them to program variables.

INPUT\$ Returns a string of characters read from the

standard input device, the keyboard, or from a

file.

INSTR Searches for the first occurrence of a given

substring in a string, and returns the position at

which the match is found.

INT Returns the largest integer that is less than or

equal to the argument.

IOCTL Sends a "Control Data" string to a character

device driver once the device has been OPENed.

IOCTL\$ Returns a "Control Data" string from a

character device driver that is OPEN.

KEY Defines and/or displays the function key

assignment text.

Introduction

KEY(n)	Enables	disables	or	terminates	interrupts of	hazus
1713 1 (11)	minanies,	uisabies,	OI	termmates	miletiable c	auseu

by a specific key.

KILL Deletes a disk file.

LCOPY Dumps the screen text to the printer.

LEFT\$ Returns a substring extracting the leftmost

number of characters from a specified string as

specified by the "length" parameter.

LEN Returns the number of characters in a given

string.

LET Assigns a value to a variable.

LINE Draws either a line, a rectangle, or a filled

rectangle. (Graphics Mode.)

LINE INPUT Inputs an entire line (up to 254 characters) to a

string variable, without delimiters.

LINE INPUT# Reads an entire line (up to 254 characters)

without delimiters, from a sequential disk data

file to a string variable.

LIST Lists the current program to the screen or to a

specified file or device.

LLIST Lists the current program on the printer.

LOAD Loads a program into memory from a file.

LOC Returns the current position of the file.

LOCATE	In Graphics Mode, moves the graphics cursor to the specified position. In Text Mode, LOCATE moves the cursor to the specified position on the active page. In both modes, LOCATE may also turn the cursor on and off and define the size of either the overwrite or the user cursor.
LOF	Returns the length of the named file in bytes.
LOG	Returns the natural logarithm of a positive argument.
LPOS	Returns the current position of the printhead within the printer buffer.
LPRINT	Prints data on the printer.
LPRINT USING	Prints data on the printer using a specified format.
LSET	Stores a string value in a random buffer field left justified, or left justifies a string value in a string variable.
MERGE	Merges the current program with another program previously saved in ASCII format.
MID\$	As a function, MID\$ returns a substring from a specified string. As a statement, replaces a portion of one string with another string.
MKDIR	Makes a new directory on a specified disk.
MKD\$	Converts a double-precision number to an eight-byte string.

MKI\$	Converts an integer to a two-byte string.
MKS\$	Converts a single-precision number to a four-byte string.
NAME	Changes the name of a disk file.
NEW	Deletes the current program and clears all variables, so that you can enter a new program.
OCT\$	A function that returns a string that is the octal value of the decimal argument.
ON COM(n) GOSUB	Specifies the first line number of a trap routine to be activated as soon as characters arrive in the communications buffer.
ON ERROR GOTO	Enables error trapping and specifies the first line number of a subroutine to be executed if an error occurs.
ONGOSUB and ONGOTO	ONGOSUB calls one of several specified subroutines, depending on the value of the specified expression. ONGOTO branches like ONGOSUB but does not return from the branch.
ON KEY(n) GOSUB	Specifies the first line number of a subroutine to be executed when a specified key is pressed.
ON PLAY(n) GOSUB	Specifies the first line number of a subroutine to be executed when the music buffer contains fewer than "n" notes. This permits continuous background music during program execution.
ON STRIG(n)	Specifies the first line number of a subroutine to be executed when one of the joystick buttons (triggers) is pressed.

ON TIMER(n) Causes an event trap every "n" seconds.

GOSUB

OPEN Allows I/O to a file or device.

OPEN COM Opens a communications file.

OPTION Defines the minimum value for array subscripts.

BASE

OUT Transmits a byte to an output port.

PAINT Paints an enclosed area on the screen with a

specified color. (Graphics Mode.)

PEEK Returns the byte read from the specified memory

location.

PLAY Plays music in accordance with a string that

specifies the notes to be played and the way in

which the notes are to be played.

PLAY(n) Returns the number of notes remaining in the

music background buffer.

PLAY ON/ Enables, disables, or suspends PLAY(n) trapping. OFF/STOP

PMAP Converts physical coordinates to world

coordinates or vice versa. (Graphics Mode.)

POINT With two arguments (x,y), returns the color

number of a pixel on the screen. If one argument(n) is given, returns the current graphics coordinate. (Graphics Mode.)

POKE Writes a byte into a memory location.

Introduction

POS Returns the current horizontal (column) position

of the cursor.

PRESET Draws a point at the specified position on the

screen. (Graphics Mode.)

PRINT Outputs data on the screen.

PRINT Outputs data to the screen using a specified

USING format.

PRINT# Writes data sequentially to a disk file.

PRINT# Writes data sequentially to a disk file using a

USING specified format.

PSET Illuminates a pixel at a specified position on the

screen. (Graphics Mode.)

PUT Writes a specified number of bytes to a

(COM files) communications file.

PUT(Files) Writes a record from a random buffer to a

random file.

PUT(Graphics) Transfers the graphics image stored in an array

to the screen.

RANDOMIZE Reseeds the random number generator.

READ Reads values from one or more data statements

and assigns them to variables.

REM Allows explanatory remarks to be inserted in a

program.

RENUM Changes the line numbers of the current

program.

RESET Closes all open data files on all drives.

RESTORE Permits DATA statements to be re-read either

from the beginning of the internal data or from a

specified file.

RESUME Continues program execution after an error trap-

ping routine has been performed.

RIGHT\$ Returns a substring from a specified string,

extracting the rightmost characters as specified

by the "length" parameter.

RMDIR Removes an existing directory.

RND Returns a random number between 0 and 1.

RSET Stores a string value in a random buffer field

right justified, or right justifies a string value in

a string variable.

RUN Runs the current program or loads a program

from disk and runs it.

SAVE Saves the current program on disk.

SCREEN The SCREEN function returns the ASCII code

(0-255) or the color number for the character at the specified row and column. The SCREEN statement sets the screen attributes that will be

used by subsequent statements.

SGN Returns 1 if the argument is positive, 0 if the

argument is zero, and -1 if the argument is

negative.

SIN Calculates the sine of the argument.

SOUND Produces a sound on the speaker.

SPACE\$ Returns a string of a specified number of spaces.

Introduction

SPC	Skips "n" spaces in a PRINT, LPRINT, or PRINT# statement.
SQR	Returns the square root of a positive expression.
STICK	Returns the x and y coordinates of two joysticks.
STOP	Terminates program execution and returns.
STRIG	Returns the status of the joystick buttons (triggers).
STRIG(n)	Enables and disables trapping of the joystick buttons.
STR\$	Returns the string representation of the value of a specified numeric expression.
STRING\$	Returns a string of specified length whose characters all have the same ASCII code or equal the first character of a given string.
SWAP	Exchanges the values of two variables.
SYSTEM	Closes all open data files and returns to MS-DOS.
TAB	Tabs the cursor or the printhead to a specified position in PRINT, LPRINT or PRINT# statements.
TAN	Returns the tangent of the argument.
TIME\$	The TIME\$ statement sets the current time. The TIME\$ function retrieves the current time.
TIMER	Returns a single precision number indicating the seconds that have elapsed since midnight or system reset.

TIMER ON/	Enables, disables, or suspends	event trapping.
OFF/STOP	-	

TROFF (Trace Off) Stops the line number listing initiated

by TRON.

TRON (Trace On) Causes the line number of each state-

ment executed to be listed.

USR Calls a machine language subroutine.

VAL Converts the string expression of a number to its

numeric value.

VARPTR Returns the memory address of a variable or file

control block.

VARPTR\$ Returns the starting address of the file control

block for a specified file.

VIEW Defines subsets of the screen called "viewports."

VIEW PRINT Sets the boundary of the text window.

WAIT Suspends program execution while monitoring

the status of a machine input port.

WHILE... Loops through a series of statements as long as a

WEND given condition remains true.

WIDTH Sets the line width in characters.

WINDOW Permits the redefinition of the screen coordinates.

(Graphics Mode.)

WRITE Writes data to the screen.

WRITE# Writes data to a sequential file.

ABS Function

Returns the absolute value of a numeric

expression.

Syntax ABS (numexp)

Remarks The returned value will always be positive or

zero.

Example Ok

PRINT ABS(8*(-6))

48 Ok Returns the ASCII decimal code for the first character of a given string.

Syntax

ASC (stringexp)

Remarks

The ASC function returns the ASCII code (0-255) corresponding to the first character of the string expression. See Appendix A for a complete list of all ASCII codes.

If "stringexp" is null, an "Illegal function call" error is returned.

See the CHR\$ function, later in thic chapter, for ASCII-to-string conversion. CHR\$ is the inverse of the ASC function.

Example

The following example shows that the ASCII code for capital letter "T" is 84.

10 X\$ = "TEST"
20 PRINT ASC(X\$)
RUN
84
Ok

ATN Function

Returns the arctangent of the argument.

Syntax

ATN (numexp)

Remarks

The evaluation of ATN is performed in single precision, unless you specify /D, (double precision), when you invoke GWBASIC.

The result is expressed in radians and falls in the range -PI/2 to PI/2 (where PI = 3.141593).

Example

10 INPUT X 20 PRINT ATN[X] RUN ? 3 1.249046

Ok

Generates a line number after every carriage return. AUTO is used only for entering programs.

Syntax

AUTO [linenum][,[increment]]

linenum

is the line number used to commence numbering lines. A period may be used to indicate the current line.

increment

is the value added to a line number to produce the next line number.

Remarks

AUTO begins numbering at "linenum" and increments each subsequent line number by "increment." The default for both values is 10. If "linenum" is followed by a comma but "increment" is not specified, the last increment specified in an AUTO command is assumed. If "linenum" is omitted but "increment" is included, line numbering begins with 0.

If AUTO generates a line number that is already being used, an asterisk is displayed after the number to warn you that any input will overwrite the existing line. Typing a carriage return immediately after the asterisk saves the line and generates the next line number.

AUTO Command

AUTO is terminated by pressing CTRL and BREAK. The line in which CTRL and BREAK is pressed is not saved. After CTRL and Break is pressed, GWBASIC returns to command level.

Examples

AUTO

Generates line numbers 10, 20, 30, 40...

AUTO 100,20

Generates line numbers 100, 120, 140...

AUTO 200,

Generates line numbers 200, 220, 240, 260, . . . The increment is 20 because 20 was the increment in the last AUTO command.

AUTO, 15

Generates line numbers 0,15,30,45,...

Activates the bell.

Syntax BEEP

Remarks In the following example, the program is

checked to see if "X" is out of the accepted range. MIN and MAX are variables containing

the limits of the accepled range.

PRINT CHR\$(7); sends an ASCII BEL character, which also activates the bell.

Example 2430 IF (X < MIN) or (X > MAX) THEN BEEP



Loads a memory image file into memory.

Syntax

BLOAD {filename} [, offset]

filename

is a string expression that specifies the file to be loaded. If the device is omitted, the MS-DOS default drive is assumed.

offset

is an integer expression in the range 0 to 65535. This is the offset into the segment declared by the last DEF SEG statement at which loading is to start.

Remarks

The BLOAD statement allows you to load assembly language routines into memory. When these routines are resident in memory, they can be CALLed from your GWBASIC program by a CALL statement.

The BLOAD and BSAVE statements allow you to load and save any portion of memory. For example, you can save and display screen images (specifying the screen buffer as the current segment by a DEF SEG statement).

If "offset" is omitted, the offset specified at BSAVE is assumed and the file is loaded into the same location from which it was saved.

If "offset" is specified, a DEF SEG statement should be executed before the BLOAD. When "offset" is given, GWBASIC assumes you want to BLOAD at an address other than the one saved. The last known DEF SEG address will be used. If no DEF SEG statement has been given, the GWBASIC data segment is used as the default (because it is the default for DEF SEG).

Warning

BLOAD does not perform an address range check. It is possible to load a file anywhere in memory. Be careful not to load over GWBASIC or the operating system.

Example

- 10 'Load a machine language program
- 20 'into memory at 60:F000
- 30 'Restore Segment to GWBASIC's DS.
- **40 DEF SEG**
- 50 'Load PROG1 into the DS.
- 60 BLOAD "B:PROG1",&HF000

Example

- 10 'Load the screen buffer
- 20 'Point segment at screen buffer
- 30 DEF SEG = &HB800
- 40 'Load FILE1 into screen buffer
- 50 BLOAD "FILE1",0

Note the DEF SEG statement in 30 and the offset of 0 in 50: this guarantees that the correct address is used.



Saves sections of the main memory on the specified file.

Syntax BSAVE {filename | pathname}

offset, length

filename is a string expression which specifies the name of the file to

be saved, with an optional device. If the device is omitted,

the MS-DOS default drive is assumed.

offset is an integer expression in the range 0 to 65535. This is the

offset into the segment declared by the last DEF SEG.

Saving starts at this position.

length is an integer expression in the range 1 to 65535, specifying

the length of the memory image to be saved.

Remarks A memory image file is a byte-for-byte copy of

what is in memory.

BSAVE saves assembly language programs

on diskette.

The BLOAD and BSAVE statements also allow you to load and save any portion of memory. For instance, you can save and display screen images (specifying the screen buffer as the current segment by a DEF SEG statement).

A DEF SEG statement should be executed before the BSAVE. The last known DEF SEG address is always used for the save.

Example

10 'Save PROG1 20 DEF SEG = &H6000 30 BSAVE "PROG1",&HF000,256

This example saves 256 bytes starting at 6000:F000 in the file "PROG1."

Example

10 'Save the screen buffer 20 'Point segment at screen buffer 30 DEF SEG=&HB800 40 'Save screen buffer in FILE1 50 BSAVE ''A:FILE1'',0,16384

The DEF SEG statement must be used to set up the segment address to the screen buffer. The offset of 0 and the length 16384 specify that the entire 16K screen buffer is to be saved.

Note: The above example will not work to save screens created with the "SCREEN 100" mode. For "SCREEN 100" screens, save 32K (32767) bytes.



Transfers control to a machine language subroutine.

Syntax CALL numvar [(variable [,variable]...)]

numvar is a numeric variable. It must equate to the starting memory

offset address of the assembly routine. The address is an offset into the current memory segment as set by the last

DEF SEG statement.

variable is a numeric or string variable which serves as an argument

to pass data between the main program and the assembly

routine.

Remarks The CALL statement is one way to transfer

program flow to an external subroutine. You can also transfer control to an external

subroutine with the USR function.

Example **110 MYROUT = &HD000**

120 CALL MYROUT (I,J,K)

The CALLS statement is the same as the CALL statement with the exceptions given below under "Remarks."

Syntax

CALLS numvar [(variable], variable]...)]

numvar

is a numeric variable. It contains the address that is the starting point in memory of the subroutine being CALLed

variable

is a numeric or string variable which may be passed as an argument to the machine language subroutine.

Remarks

The CALLS statement is similar to CALL, except that the segmented addresses of all arguments are passed. (CALL passes unsegmented addresses.) CALLS should be used when accessing routines written with FORTRAN calling conventions, since all FORTRAN parameters are call-by-reference segmented addresses.

CALLS uses the segment address defined by the most recently executed DEF SEG statement to locate the routine being called.

CDBL Function

Converts a given numeric expression to a double precision number.

Syntax

CDBL (numexp)

Example

10 A=454.67 20 PRINT A;CDBL(A) RUN 454.67 454.6700134277344 Ok

Note: A and CDBL (A) do not have precisely the same value due to the difference in the way that single and double precision numbers are stored internally. For more information, see the Appendix on Advanced Features. Transfers control and passes variables to another program.

Syntax CHAIN [MERGE] filename[, [linenum]

[, [ALL] [,DELETE range]]]

filename is a string expression which specifies the name of the called program file and optionally the drive. If the drive is omitted

the MS-DOS default drive is assumed.

linenum is a line number or an expression that evaluates to a line number in the called program. It is the starting point for execution of the called program. If it is omitted, execution

begins at the first line. The parameter is not affected by a RENUM command.

range the range of line numbers to be deleted if the delete option is

used.

Remarks

If the Merge option is used, a MERGE operation is performed with the current program and the CHAINed program. The CHAINed program must be an ASCII file. If any lines in the disk file have the same line numbers as lines in the program in memory, the lines from the file on disk will replace the corresponding lines in memory. (MERGEing may be thought of as "inserting" the program lines on disk into the program in memory). The MERGE option leaves the files open, preserves the current OPTION BASE setting, and preserves variable types and user-defined functions, for use by the CHAINed program.

User-defined functions should be placed before any CHAIN MERGE statements in the program. Otherwise, the user-defined functions will be undefined after the merge is complete.

If the MERGE option is omitted, the CHAINing program is lost (except common variables) before loading the CHAINed program. CHAIN does not preserve variable types or user functions. Thus, any DEFtype or DEF FN statements containing shared variables must be repeated in the CHAINed program.

If the ALL option is used, every variable in the current program is passed to the CHAINed program.

If the ALL option is used and 'linenum is omitted, a comma must hold the place of 'linenum.' For example:

100 CHAIN "NEXTPROG",,ALL

is correct, but:

100 CHAIN "NEXTPROG", ALL

is incorrect. In this case, GWBASIC assumes that ALL is a variable name and evaluates it as a line number.

If the ALL option is omitted, the current program must contain one or more COMMON statements to list the variables that are passed. (See the COMMON statement in this chapter.)

If the DELETE option is used, a section of the current program (specified by a 'range' of line numbers) will be deleted before loading the CHAINed program.

DELETE is often used with MERGE and 'line' options, to load overlays. After an overlay is brought in, it is usually desirable to delete it so a new overlay may be brought in.

Note: Before running a CHAINed program, CHAIN carries out a RESTORE. This resets the pointer to the beginning of the internal Data statements.

10 ' THIS PROGRAM DEMONSTRATES Example 1 20 ' CHAINING USING COMMON **30 ' TO PASS VARIABLES** 40 ' SAVE THIS MODULE ON DISK 50 ' AS "PROG1" WITH THE A OPTION. 60 DIM A\$(2).B\$(2) 70 COMMON AS[],BS[] 80 A\$(1)="COMMON VARIABLES NEED" 90 A\$(2) = "VALUES BEFORE CHAINING." 100 B\$(1) = " ": B\$(2) = " " 110 CHAIN "PROG2" 120 PRINT: PRINT B\$(1): PRINT 125 PRINT B\$(2): PRINT **130 END** 10 ' THE STATEMENT "DIM A\$(2),B\$(2)" 20 ' MAY ONLY BE EXECUTED ONCE. **30 ' HENCE, IT DOES NOT APPEAR** 40 ' IN THIS MODULE. 50 ' SAVE THIS MODULE ON THE DISK 60 ' AS "PROG2" USING THE A 70 ' OPTION. 80 COMMON A\$(),B\$() 90 PRINT: PRINT A\$(1);A\$(2) 100 B\$(1) = "CHAIN TO LINE 90" 110 B\$(2) = "TO SKIP DIM" 120 CHAIN "PROG1".90

130 END

Example 2

10 ' THIS PROGRAM DEMONSTRATES

20 ' CHAINING USING THE MERGE

30 ' AND ALL OPTIONS.

40 ' SAVE THIS MODULE ON THE DISK

50 ' AS "MAINPRG".

60 A\$ = "MAINPRG"

70 CHAIN MERGE "OVRLAY1",1010,ALL

80 END

1000 'SAVE THIS MODULE ON THE DISK

1010 ' AS "OVRLAY1" USING THE A

1015 ' OPTION.

1020 PRINT AS; " HAS CHAINED TO"

1025 PRINT "OVRLAY1."

1030 A\$ = "OVRLAY1"

1040 B\$ = "OVRLAY2"

1050 CHAIN MERGE "OVRLAY2",1010,

ALL, DELETE 1000-1050

1060 END

1000 'SAVE THIS MODULE ON THE DISK

1010 'AS "OVRLAY2" USING THE A

1015 ' OPTION.

1020 PRINT AS; "HAS CHAINED"

1025 PRINT " TO ";BS; " . "

1030 END

CHDIR Command

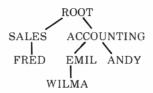
Changes the current directory.

Syntax

CHDIR pathname

pathname

is a string expression identifying the new directory which is to be the current directory



Assuming that the diskette in drive B has the directory structure illustrated above, to change the current directory from ROOT to ACCOUNTING enter:

CHDIR "B:\ACCOUNTING"

ACCOUNTING is now the current directory on drive B. To change the current directory from ACCOUNTING to ANDY enter:

CHDIR "ANDY"

Returns a one-character string whose ASCII decimal code is the value of the argument.

Syntax CHR\$ (n)

n

is an integer expression which must be in the range 0 to 255. It represents an ASCII code. If it is outside the specified range, an "Illegal Function Call" is returned.

Remarks

CHR\$ is normally used to send a special character to an output device. For instance, the BEL character (CHR\$(7)) could be sent as a preface to an error message, or a form feed (CHR\$(12)) could be sent to clear a terminal screen and return the cursor to the home position.

Examples

100 PRINT CHR\$(7) 'BEEP

150 PRINT CHR\$(LINEFEED%)

200 IF CHR\$(INP(IN.PORT%)) = "A" THEN 210 GOSUB 100

CINT Function

Converts any numeric argument to an integer by rounding the fractional portion.

Syntax

CINT (numexp)

Remarks

If "numexp" is not in the range -32768 to 32767, an "Overflow" error occurs. If the fractional portion of "numexp" is >=.5 the integer part is rounded up; otherwise a truncation occurs.

See the CDBL and CSNG functions for details on converting numbers to the double precision and single precision data type, respectively. See also the FIX and INT functions, both of which return integers.

Example

Ok PRINT CINT(45.67) 46

Ok PRINT CINT(-3.71) -4 Ok Draws a circle (or an ellipse) with the specified center and radius (Graphics mode only).

Syntax

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

x,y

are numeric expressions, specifying the coordinates of the center of the circle (or ellipse). They may be given in absolute form, or in relative form if STEP is included.

radius

is a numeric expression returning a positive integer value. It is the radius of the circle, or the major axis of the ellipse. It is measured in pixels in the horizontal direction if aspect < 1, and in vertical direction if aspect > 1.

color

is an integer expression in the range 0 to 3. It is the color number of the circumference (or ellipse). See the COLOR graphics statement (current screen mode) for details.

start.end

are numeric expressions specifying angles in radians. The range is from -2*PI to 2*PI, where PI = 3.141593. They specify where the drawing of the circle (or ellipse) will begin and end.

aspect

is a numeric expression returning a positive value. Due to the nonuniform distribution of the pixels on the screen, you must specify a value of 'aspect' to draw a true circle with different monitors. The default value of 'aspect' is 5/6 in medium and super resolution and 5/12 in high resolution. This value produces a circle with the standard monitor.

Drawing Circles and Ellipses

The CIRCLE statement draws circles if you do not specify the "aspect" parameter, and ellipses if you specify a value of "aspect" different from the default value (5/6 in medium and super resolution, and 5/12 in high resolution).

The "aspect" may be thought of as a fraction, with a separate numerator and denominator. The numerator tells GWBASIC how many rows the CIRCLE statement should consider equivalent to the number of columns specified by the denominator.

If "aspect" is less than one, then "radius" is measured in pixels in the horizontal direction, i.e., it is the x-radius. In this case GWBASIC draws ellipses with the same width, and varies the height.

If "aspect" is greater than one, the y-radius is given, and GWBASIC draws ellipses with the same height and varies the width.

For example:

100 CIRCLE (100,150),50,,,,5/18

will draw a horizontal ellipse with an x-radius of 50 pixels.

Drawing Arcs

The CIRCLE statement can simply draw part of a circle (or ellipse), i.e., an "arc."

To draw an arc you must enter the "start" and "end" parameters. They specify the first and the second arc endpoint in radians.

The angles are positioned in the standard mathematical way, with 0 to the right and going counterclockwise.

For example, the following statement specifies just a quarter of a circle:

10 CIRCLE (100,150),50,1,0,3.141593/2

The angles must be measured in radians. If you have the angles in degrees, you must convert them to radians before executing the CIRCLE statement. To convert from degrees to radians, multiply by 0.0174532.

Drawing Rays

The CIRCLE statement can draw a ray from the center of the arc to either arc endpoint. A negative endpoint generates a ray to that endpoint. The endpoint, -0, is not treated as a negative endpoint. To circumvent this limitation, use a small negative number (e.g., -0.001 instead of -0). When both endpoints are negative, both rays are drawn. The minus sign does not affect the arc itself, i.e., the angles will be treated as if they were positive. Note that this is different from adding 2*PI (where PI is 3.141593). The start angle may be greater or less than the end angle. For example:

100 CIRCLE (100,150),50,1, -0.001, -3.141593/2

will draw a quarter of a circle delimited by two rays.

Last Point Referenced

The last point referenced after a circle (or ellipse) has been drawn is the center of the circle (or ellipse).

Clipping

Points that are off the screen or the graphics viewport are not drawn by the CIRCLE statement.

STEP Option

Coordinates can be shown as absolutes or the STEP option can be used to reference a point relative to the most recent point used.

For example, if the most recent point referenced was 100,50, then:

either

CIRCLE (200,200),50

or

CIRCLE STEP (100,150),50

will draw a circle at 200,200 with radius 50. The first example uses absolute notation; the second uses relative notation.

Example

The following example draws three intersecting circles and colors the area of intersection.

5 SCREEN 1 10 COLOR 0,3 20 CLS 30 CIRCLE (100,120),90 40 CIRCLE (150,130),120 50 CIRCLE (250,120),100 60 PAINT (180,120)

CLEAR Command

Clears all numeric variables to zero, all string variables to null, and closes all open files. Options set the highest memory location available for use by GWBASIC, and the amount of stack space.

Syntax

CLEAR [, [memory] [, stack]]

memory

is an expression representing a memory location which, if specified, sets the top of memory (i.e., the maximum extension of the GWBASIC Data Segment)

stack

is an integer expression whose value sets aside stack space for GWBASIC. The default is 512 bytes or one-eighth of the available memory, whichever is smaller.

Remarks

The "memory" parameter should be specified to reserve space in storage for assembly language routines, the "stack" parameter to use several nested GOSUBs, FOR...NEXT loops, or PAINT to paint complex pictures.

GWBASIC allocates string space dynamically. An "Out of string space" error occurs only if there is no free memory left for GWBASIC to use. If a value of 0 is given for either expression, the appropriate default is used. The default stack size is 512 bytes, and the default top of memory is the current top of memory. The CLEAR statement performs the following actions:

- closes all files
- clears all COMMON variables
- resets the stack and string space
- resets all simple numeric variables and numeric array elements to zero
- resets all simple string variables and string array elements to null
- releases all disk buffers
- resets all DEF FN, DEFINT/SNG/DBL/STR, DEF SEG and DEF USR statements

Examples

CLEAR

CLEAR ,32768

CLEAR ,,2000

CLEAR ,32768,2000



Terminates I/O to a file or device. CLOSE is usually used in a program.

Syntax

CLOSE [[#] filenum[, [#]filenum]...]

filenum

is the number under which the file was OPENed. A CLOSE with no arguments closes all open files.

Remarks

The association between a particular file and file number terminates upon execution of a CLOSE statement. The file may then be reOPENed using the same or a different file number; likewise, the file number may now be reused to OPEN any file.

A CLOSE for a sequential output file writes the final buffer of output.

The END statement and the NEW command always close all disk files automatically. (STOP does not close disk files.)

Example

To read the data in a sequential file open for output or append, you must first CLOSE the file and then reOPEN it in the "I" mode. If the file DATA was opened for output as #1:

100 CLOSE #1 110 OPEN "I", #1, "DATA" Erases all or part of the screen.

Syntax

CLS [n]

n

is an integer expression in the range 0 to 2.

Remarks

CLS without a parameter clears the entire screen to the current text background color, unless a graphics viewport has been defined, and resets the function key line (if the function key display is enabled). If a viewport has been defined, the current viewport only will be cleared to the graphics background color. Outputting a formfeed character (typing CTRL L or PRINT CHR\$(12);, will have the same effect).

CLS 0 clears the entire screen, resetting the function key display.

CLS 1 clears the graphics viewport to the graphics background color (in one of the graphics modes). If no viewport has been defined, this will have no effect.

CLS 2 clears the text window to the text background color, without resetting the function key display.

CLS not only erases all or part of the screen, but also returns the cursor to the upper lefthand corner of the screen (in Text Mode).

If you are in Graphics Mode, CLS makes the "last referenced point" the center of the screen.

The screen can also be cleared by pressing CTRL HOME or by modifying the screen mode using the SCREEN statement, or the width using the WIDTH statement.

Example

10	CLS	' Clears the screen (or
20		' the current viewport]
60	CLS 0	' Clears whole screen
90	CLS 1	' Clears the graphics
100		' viewport
110	CLS 2	' Clears the text window

Sets the text foreground and background colors (Text Mode only).

Syntax

COLOR [foreground] [,background] [,dummy]

foreground

is a numeric expression rounded to the nearest integer. It must be in the range 0 to 31. It selects the character foreground color.

background

is a numeric expression rounded to the nearest integer. It must be in the range 0 to 15, but it is interpreted modulo 8, thus only values from 0 to 7 are taken into consideration.

dummy

this parameter is allowed for compatibility with other BASICs. It will have no effect. It may specify border color on other systems.

Remarks

(Color Text Mode)

If you enable color (see the SCREEN statement) or the color hardware is installed (Standard monitor), the following colors are allowed for "foreground":

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

To make characters blink for a specific color, you should set "foreground" equal to 16 plus the color number.

You may select only colors 0 through 7 for "background."

In a monochrome system the following values can be used for "foreground":

- 0 Black
- 1 Underlined character with white foreground
- 2-7 White

Adding 8 to the number of the desired color gives you the color in high-intensity.

You can make the character blink by adding 16 to the number of the desired color.

For "background" you may select the following values:

0-6 Black 7 White

Foreground and background colors may be equal. In this case any character displayed is invisible. Changing the foreground or background color will make subsequent characters visible again.

Any parameter may be omitted. Omitted parameters assume the old value.

Upon initialization, the default values are:

- foreground = 7 (White)
- background = 0 (Black)

That is, if no COLOR statement exists in your program, the system assumes:

COLOR 7,0

Examples

100 COLOR 0,2

This sets a black foreground on a green background in color mode and a black foreground on a black background, i.e., invisible characters, in B/W mode.

150 COLOR 15,1

This sets a high-intensity white on a blue background in color mode, and a high intensity white on a black background in B/W mode.

Possible Errors

If the COLOR statement ends in a comma (,), a "Missing operand" error is returned, but the color will change. For example:

COLOR 2,

is invalid.

If you enter a value outside the range 0 to 255 an "Illegal function call" error is returned. Previous values are retained.

Medium-resolution Graphics

Defines the background and foreground pallette colors.

Syntax

COLOR [background] [,palette]

background

is a numeric expression rounded to the nearest integer. It must be in the range 0 to 15. It selects color for the character background. It defaults to 0 (Black) if unspecified.

palette

is a numeric expression rounded to the nearest integer. It must be in the range 0 through 255. This selects one of 2 palettes for the color numbers 1, 2 and 3 that may be specified in a graphics statement.

Palette	Color 1	Color 2	Color3
0	Green	Red	Yellow
1	Cyan	Magenta	White

Palette 0 is selected, when 'palette' is an even number. Palette 1 is selected, if 'palette' is an odd number.

Remarks

When you enter a CIRCLE, DRAW, LINE, PAINT, PRESET, or PSET statement in your program, you can specify a color number of 0, 1, 2, or 3. This parameter selects the color from the current "palette" as defined by the COLOR statement.

If you do not specify a color number, the default is color 3.

When you display text the character foreground will be color number 3. The character background will be set by the color statement.

Any parameter may be omitted in the COLOR statement. Omitted parameters assume the old value.

Upon initialization the default values are:

- background =0
- palette =1

That is, if no COLOR statement exists in your program, the system assumes:

COLOR 0, 1

The use of memory for color and monochrome in medium-resolution mode is identical. The modes differ only in that the two bits of a pixel are interpreted differently by the hardware: B/W medium resolution displays 4 shades of grey.

Examples

10 SCREEN 1,0 20 COLOR 10,1

Sets the palette background to light green, and selects palette 1 (Cyan, Magenta, White).

100 COLOR,0

The background stays light green and palette 0 is selected.

High and Super-resolution Graphics

Defines the graphics and foreground text colors.

Syntax COLOR [foreground] [, dummy]

foreground is a numeric expression rounded to the nearest integer. It must be in the range 0 to 15. This specifies both graphics and

text foreground color and defaults to White (color 7). The

background color is always black.

dummy is ignored in this mode.

Remarks

When you enter a CIRCLE, DRAW, LINE,
PAINT, PRESET, or PSET statement in your
program, you can specify a color number of 0,
1, 2, or 3. A color of 0 or 2 indicates black. A

color of 1 or 3 selects the color from the current foreground as defined by the COLOR statement.

If you do not specify a color number, the default is the last foreground color specified.

Any parameter in the COLOR statement may be omitted. Omitted parameters assume the old values. Upon initialization default values are:

• foreground = 7(White)

That is, if no COLOR statement exists in your program, the system assumes:

COLOR 7

Example SC

SCREEN 2 COLOR 14

This selects a yellow foreground on a black background.

Enables or disables event trapping of communications activity on the specified channel.

Syntax COM (n) {ON | OFF | STOP}

n is an integer expression that specifies the number of the communications channel. It may be 1, 2, 3, or 4.

COM(n) ON

enables communications event trapping. While trapping is enabled, and if a non-zero line number is specified in the ON COM(n) GOSUB statement, GWBASIC checks between every statement to see if activity has occurred on the

communications channel. If it has, the ON COM(n) GOSUB statement is executed.

COM(n) OFF disables communications event trapping.

COM(n) STOP disables communications event trapping, but if an event occurs it is remembered, and ON COM(n) will be executed as

soon as trapping is enabled.

Example

10 COM(1) ON

Enables error trapping of communications activity on channel 1.

Defines a common area which is not erased by the CHAINed program, and allows you to pass variables from one program to another.

Syntax

COMMON variable [,variable]...

variable

is the name of a numeric or string variable which is required to be passed to the CHAINed program. For array variables place a set of parentheses "()" after the variable name.

Remarks

The COMMON statement is used in conjunction with the CHAIN statement. COMMON statements may appear anywhere in a program, though it is recommended that they appear at the beginning.

The CHAINed program need not specify, through the use of COMMON statements, the common variables specified by the CHAINing program. The CHAINed program will use those variables with the same names specified in the CHAINing program. Each type definition statement (DEFINT, DEFSNG, DEFDBL, DEFSTR) referring to common variables, must precede the associated COMMON statements and must be repeated in the CHAINed program.

Common variables must always be initialized within the CHAINing program. Common arrays must be explicitly described by DIM statements in the CHAINing program (but not in the CHAINed program, otherwise a "Duplicate definition" error occurs). The DIM statements must be written before the associated COMMON statements.

Example

10 REM PG1 20 COMMON A1,B1,C1,D1\$

80 CHAIN "A:PG2" 90 END

10 REM PG2 20 PRINT A1,B1,C1,D1\$ 120 END

The above example shows that the CHAINed program need not specify, through the use of COMMON statements, the common variables specified by the CHAINing program.

In our example the values of the variables A1, B1, C1, and D1\$ in the program PG1 are passed to the CHAINed program PG2, which displays them.

The DIM statement must be written before the associated COMMON statement.

Example

10 REM PG1 20 DEFDBL C1 30 COMMON A1,B1,C1,D1\$

90 CHAIN "A:PG2" 100 END

10 REM PG2 20 DEFDBL C1

130 END

Each type definition statement (DEFINT, DEFSNG, DEFDBL, DEFSTR) referring to common variables, must precede the associated COMMON statement and must be repeated in the CHAINed program. (Note the statements DEFDBL, both with PG1 and PG2.)

Example

10 REM PG1 20 DIM A1(15,20) 30 COMMON A1(),B1,C1

:

. 100 CHAIN "A:PG2" 110 END

10 REM PG2

•

•

50 PRINT A1(1,1)

.

90 END

A COMMON statement can also specify array names. Such specifications are followed by a pair of parentheses.

Each use of common array must be explicitly described by a DIM statement in the CHAINing program (but not in the CHAINed one, otherwise a "Duplicated Definition" error occurs).

The DIM statement must be written before the associated COMMON statement.

Example

10 REM mod1 20 A = 1:B = 2 30 COMMON A,B 50 COMMON C 60 CHAIN "mod3"

10 REM mod2 20 A = 1:B = 2 30 COMMON A 40 GOTO 60 50 COMMON B 60 CHAIN "mod3"

10 REM mod3 20 PRINT A;B

For example, when executing program "mod1" an "Illegal function call in 50" is issued, as variable C has not been initialized. When executing program "mod2" instead, program "mod3" is CHAINed: it displays both A and B variables, even if statement 50 of "mod2" is jumped over.

Resumes program execution after a CTRL-BREAK has been typed or a STOP or END statement has been executed. CONT should only be used in immediate mode.

Syntax CONT

Remarks

Execution resumes at the point where the break occurred. If the break occurred after a prompt from an INPUT statement, execution continues with the reprinting of the prompt ("?" or prompt string).

CONT is usually used in conjunction with STOP for debugging. When execution is stopped, intermediate values may be examined and changed using direct mode statements. Execution may be resumed with CONT or a direct mode GOTO, which resumes execution at a specified line number.

CONT may not be used to continue execution after an error has occurred. CONT is also invalid if the program has been modified during the break.

Example

```
10 INPUT A,B
20 TEMP = A*B
30 STOP
40 FINAL = TEMP + 300: PRINT FINAL
RUN
? 32, 2.4
Break in 30
Ok
PRINT TEMP
76.8
Ok
CONT
376.8
Ok
```

Returns the cosine of the argument.

Syntax COS (numexp)

Remarks The argument "numexp" represents the angle

in radians.

The calculation of the COS function is performed in single precision, unless "/D" is supplied in the GWBASIC command line.

Example

10 X = 2 * COS(.4) 20 PRINT X RUN 1.842122 Ok



Converts any numeric argument to a single

precision number.

Syntax CSNG (numexp)

Remarks See the CINT and CDBL functions for

converting numbers to the integer and double

precision data types, respectively.

Example

10 A# = 975.342123217685 20 PRINT A#; CSNG(A#) RUN

975.342123217685 975.3421

Ok

Returns the current line (row) position of the cursor.

Syntax CSRLIN

Remarks CSRLIN returns a value in the range 1 to 25.

To return the current column position use the POS function. (See the POS function in this

chapter.)

Example

10 Y = CSRLIN 20 X = POS(0) 30 LOCATE 24,1 35 REM PRINT 40 LOCATE Y,X 'Record current line.
'Record current column.

:Print "HELLO" HELLO on last line. 'Restore position to old

line, column.

Converts string values to numeric values.

Syntax 1 CVI(2-byte-string)

Syntax 2 CVS(4-byte-string)

Syntax 3 CVD(8-byte-string)

Remarks Numeric values that are read in from a random file buffer must be converted from strings back into numbers.

CVI converts a "2-byte-string" to an integer.

CVS converts a "4-byte-string" to a single precision number.

CVD converts an "8-byte-string" to a double precision number.

See also "MKI\$, MKS\$, MKD\$" functions, later in this chapter.

Example

70 FIELD #1,4 AS N\$, 12 AS B\$ 80 GET #1 90 Y=CVS(N\$) Creates an "internal" file, i.e., a sequence of data belonging to the program. Each data item can then be assigned to a program variable by a READ statement. A DATA statement should only be used in a program.

Syntax

DATA constant[, constant]...

constant

is a numeric or string constant. Any numeric format (i.e., integer, hexadecimal, octal, single or double precision) is acceptable for numeric constants. String constants in DATA statements must be surrounded by double quotation marks only if they contain commas, colons, or significant leading or trailing spaces. Otherwise, quotation marks are not needed.

Remarks

DATA statements are non-executable and may be placed anywhere in the program. A DATA statement may contain as many constants as will fit on a line (separated by commas). Any number of DATA statements may be used in a program.

A DATA statement in a program need not correspond to a specific READ statement. This is because before program execution, a data file (the "internal file" as it is often called) is created. It contains all the values of all the DATA statements in the program in line number sequence. When the program is executed, READ takes its values from this file.

The data-type of an entry in the data sequence must correspond to the type of the variable to which it is to be assigned; i.e., numeric variables require numeric constants as data (conversion from one numeric type to another is allowed; for example, you may have a single precision floating point constant associated with an integer variable) and string variables require quoted or unquoted strings as data.

DATA statements may be re-read from the beginning by use of the RESTORE statement.

Example

Ok
10 PRINT "CITY", "STATE", "ZIP"
20 READ C\$,S\$,Z
30 DATA "BIRMINGHAM,"
35 DATA "ALABAMA,12345"
40 PRINT C\$,S\$,Z
RUN
CITY STATE ZIP
BIRMINGHAM, ALABAMA 12345
Ok

Retrieves the date (as a function), or sets the date (as a statement).

Syntax1

stringvar = DATE\$
Used as a function

Syntax2

DATE\$ = stringexp Used as a statement

Remarks

As a function, the current date is fetched and assigned to the string variable "stringvar" The DATE\$ function may also be used in any string expression in a LET or PRINT statement.

As a statement, the current date is set. In this case DATE\$ is the target of a string assignment.

The date may also have been set by MS-DOS prior to entering GWBASIC.

Rules

- If "stringexp" is not a valid string, a "Type Mismatch" error will result. Previous values are retained.
- For "stringvar" = DATE\$, DATE\$ returns a 10 character string in the form "mm-dd-yyyy" where mm is the month (01 to 12), dd is the day (01 to 31) and yyyy is the year (1984 to 1991).

• For DATE\$ = "stringexp," "stringexp" may be one of the following forms:

```
"mm-dd-yy"
or
"mm/dd/yy"
or
"mm-dd-yyyy"
or
"mm/dd/yyyy"
```

If the month or day is specified by the use of only one digit, GWBASIC assumes a 0 (zero) in front of it. If the year is specified by the use of one digit (y), GWBASIC assumes the year to be 200y; if two digits are specified (yy), the year will be 19yy.

If any of the values are out of range or missing, an "Illegal function call" error is issued. Any previous date is retained.

Example

DATES = "01-01-84"
Ok
PRINT DATES
01-01-1984
Ok

Defines and names user-written function. A DEF FN statement may only be used in a program.

Syntax

DEF FN name[(argument[,argument]...)] = expression

name

a legal variable name beginning with FN. No blanks may be inserted between FN and the remainder of the name and the first character after FN must be a letter.

argument

a "dummy" variable that is to be replaced by the corresponding argument value when the function is called.

expression

an expression that performs the operation of the function. The type of expression must agree with the type (numeric or string) of the function, specified by "name".

Remarks

In the DEF FN statement, variable names serve only to define the function; they do not affect program variables that have the same name. A variable name used in a function definition may or may not appear in the argument list. If it does, the value of the parameter is supplied when the function is called. Otherwise, the current value of the program variable is used.

The variables in the argument list represent, on a one-to-one basis, the argument variables or values that are to be given in the function call.

User-defined functions may be numeric or string. The type of the function is specified by "name." The type of the expression must match the type of the function, otherwise a "Type Mismatch" occurs. If the function is numeric the value of the expression is forced to that type

If a DEF FN statement has not been executed before the function it defines is called, an

Example

400 R=1:S=2 410 DEF FNAB(X,Y)=X^3/Y^2 420 T=FNAB(R,S)

before the function value is returned.

"Undefined user function" error occurs.

Line 410 defines the function FNAB. The function T will contain the value (R ^3) divided by (S ^2) or .25.

Assigns the current segment of memory.

Syntax

DEF SEG [= address]

address

is a numeric expression returning an unsigned integer in the range 0 to 65535. The address specified identifies the segment address used by BLOAD, BSAVE, PEEK, POKE, DEF USR, and CALL.

If 'address' is omitted, then the segment to be used is set to GWBASIC's data segment (i.e., the beginning of your user workspace in memory). This is the initial default value.

If 'address' is specified, then it will be based upon a 16 byte boundary. For the BSAVE, PEEK, POKE, or CALL statement, the value is shifted left 4 bits to form the Code Segment address for the subsequent call instruction.

Note: GWBASIC does not check if the resultant segment is valid.

If you enter a value outside the range, then an "Illegal function call" error results. Previous value will be retained.

If you do not separate DEF and SEG by at least one blank, GWBASIC would interpret the statement:

DEFSEG = 150

to assign the value 150 to the variable DEFSEG

10 DEF SEG = &HB800 'Set segment to 15 'Screen buffer 20 DEF SEG 'Restore segment to 25 'GWBASIC's DS

Note that in statement 10 the screen buffer is at absolute address B8000 hex, as the last hexadecimal digit is dropped on the DEF SEG statement.

Enables access to a machine language subroutine by specifying the starting address. The subroutine may be subsequently called by the associated USR function. DEF USR is usually used in a program.

Syntax DEF USR [n] = offset

n may be any digit from 0 to 9. The digit corresponds to the number of the USR routine whose address is being specified.

If 'n' is omitted, DEF USR0 is assumed.

offset Offset is an integer expression in the range 0 to 65535.

Remarks

Any number of DEF USR statements may appear in a program to redefine subroutine starting addresses, thus allowing access to as many subroutines as necessary. To obtain the starting address of a subroutine, GWBASIC adds the value of "offset" to the current

segment value.

Example 100 DEF SEG = 0

200 DEF USR0 = 24000 210 X = USR0(Y^2/2.89)

DEFINT/SNG/DBL/STR

Statements

Remarks

Declare the variable type in accordance with the letter(s) specified. These statements are usually used in a program.

Syntax DEF type letter[-letter][.letter[-letter]]...

type is INT, SNG, DBL. or STR. No space should be entered between DEF and INT, SNG, DBL, or STR.

letter represents a letter from the alphabet (A-Z)

· ·

Any variable names beginning with the letter(s) specified in "range of letters" will be considered. The type of variable specified by the "type" declaration character (%,!,#,\$) always takes precedence over a DEFtype statement.

If no type declaration statements are encountered, GWBASIC assumes all variables without declaration characters are single precision variables. DEFtype statements must precede the use of the defined variables.

Example 10 DEFDBL L-P

All variables beginning with the letters L, M, N, O, and P will be double precision variables.

10 DEFSTR A

All variables beginning with the letter A will be string variables.

10 DEFINT I-N,W-Z

All variables beginning with the letters I, J, K, L, M, N, W, X, Y, Z will be integer variables.

Erases program lines. DELETE is usually used in immediate mode.

Syntax

DELETE [linenum1][- [linenum2]]

linenum1

first line to be erased.

linenum2

last line to be erased.

Remarks

GWBASIC always returns to command level after a DELETE is executed. If either "linenum1" or "linenum2" does not exist, an "Illegal function call" error occurs unless both "linenum1" and "linenum2" are used to specify a range of line numbers to be deleted. In this case it is acceptable to specify a non-existent line number for "linenum1," providing "linenum2" is a valid line number. A period (.) can be used instead of the line number to indicate the current line.

DELETE 80

Deletes line 80.

DELETE 80-120

Deletes lines 80 through 120, inclusive.

DELETE -80

Deletes all lines up to and including line 80.

DELETE 80-

Deletes all lines from line 80 through the end of the program.

Specifies the array name, the number of dimensions and the subscript upper bound per dimension. The DIM statement may specify one or more arrays.

Syntax

DIM array (subscripts)[,array (subscripts)]...

array

is a valid array name. Any legal variable name may be used.

subscripts

refers to one or more numeric expressions which specify the array dimensions. Each subscript must be separated from the next by commas. The number of subscripts specifies the number of dimensions, and the value of each specifies the subscript upper bound.

Remarks

If an array name is used without a corresponding DIM statement, the maximum value of the array's subscript(s) defaults to 10. If a subscript is used that is greater than the maximum specified, a "Subscript out of range" error occurs. The minimum value for a subscript is always 0, unless otherwise specified with the OPTION BASE statement.

If no DIM is specified, the first reference to an array element in the program will create the array with the specified number of dimensions. For example, if a program statement refers to: AR1(3,5,10) then AR1 is created with 3 dimensions and a default upper bound of 10 for each dimension.

The DIM statement sets all numeric array elements to an initial value of zero and elements of string arrays to null strings.

Theoretically, the maximum number of dimensions allowed in a DIM statement is 255 and the maximum number of elements per dimension is 32767. In reality, however, these numbers are limited by line length and memory size.

If you try to redimension an array without first erasing it, a "Duplicate definition" error occurs. You must first use the ERASE statement to erase an array before redimensioning it.

Number of Elements per Dimension

no DIM is used
OPTION BASE 0 is set 11 elements (subscripts 0-10
are allowed in each dimension)

OPTION BASE 1 is set 10 elements (subscripts 1-10 are allowed in each dimension)

DIM is used
OPTION BASE 0 is set the number of elements in
each dimension is calculated by adding 1 to each
upper bound subscript

OPTION BASE 1 is set the number of elements in each dimension coincides with each upper bound subscript

To Define an Array

- 1 Establish the subscript lower bound. Use OPTION BASE 1 or adopt the default OPTION BASE 0.
- 2 Assign a name to the array using a DIM statement.
- 3 Establish the number of dimensions using the DIM statement.
- 4 Establish the subscript upper bounds per dimension using the DIM statement.

If you do not dimension an array, its implicit dimensions are the default values described in remarks.

- a DIM statement cannot be preceded by an array reference
- a DIM statement does not set the subscript upper bound per dimension, in case it is jumped over.

Examples

10 DIM A(5),B\$(20,30,15)

10 INPUT I 20 DIM ARRAY1(I) 30 FOR K = 0 TO I 40 READ ARRAY1(K) 50 NEXT **Example 10 DIM A(20)**

20 FOR I=0 TO 20

30 READ A(I)

40 NEXT I

Example

LIST

10 I=1

20 GOTO 40

30 DIM A(50)

40 A(10) = 3

50 A(11)=45

Ok

RUN

Subscript out of range in 50

Ok

The system displays:

Subscript out of range in 50

when statement 50 is executed, as statement 30 is jumped over and an upper bound of 10 is assumed by default.

DRAW Statement

Draws an object as specified by the contents of a string expression. (Graphics Mode only.)

Syntax

DRAW stringexp

stringexp

is a string expression which defines an object which is drawn when GWBASIC executes the statement. 'Stringexp' is one or more of the movement commands below.

Remarks

The DRAW statement combines most of the capabilities of the other graphic statements into an easy-to-use object definition language called "Graphics Macro Language." A GML command is a single character or a pair of characters within the string "stringexp," optionally followed by one or more arguments.

Movement Macros

Each of the following movement commands begin movement from the current graphics position. This is usually the coordinate of the last graphics point plotted with another GML command, LINE, or PSET. The current position defaults to the center of the screen when a program is RUN.

U[n]	Move up. The number of points moved is n \ast scale factor (set by the S command below). If 'n' is omitted 1 is supplied.
D[n]	Move down. The number of points moved is n * scale factor (set by the S command below). If 'n' is omitted 1 is supplied.
L[n]	Move left. The number of points moved is n \ast scale factor (set by the S command below). If 'n' is omitted 1 is supplied.
R[n]	Move right . The number of points moved is n \ast scale factor (set by the S command below). If 'n' is omitted 1 is supplied.
E[n]	Move diagonally up and right. The number of points moved is n \ast scale factor (set by the S command below). If 'n' is omitted 1 is supplied.
F[n]	Move diagonally down and right. The number of points moved is n * scale factor (set by the S command below). If 'n' is omitted 1 is supplied.
G[n]	Move diagonally down and left. The number of points moved is n \ast scale factor (set by the S command below). If 'n' is omitted 1 is supplied.
H[n]	Move diagonally up and left. The number of points moved is n * scale factor (set by the S command below). If 'n' is omitted 1 is supplied.

Mx,y

Move absolute or relative. If 'x' is preceded by a plus (+) or minus (-), 'x' and 'y' are added to the current graphics position, and connected with the current position by a line (move relative). Otherwise, a line is drawn to point 'x,y' from the current position (move absolute).

В

Move without plotting any points. B may precede any of the above mentioned movement commands.

Ν

Move but return to original position when finished. N may precede any of the above mentioned movement commands.

Further GML Commands

An

Set angle 'n'. 'n' may range from 0 to 3, where 0 is 0 degrees, 1 is 90, 2 is 180, and 3 is 270. Figures rotated 90 or 270 degrees are scaled so that they will appear the same size as with 0 or 180 degrees on a monitor screen with the standard aspect ratio of 4/3.

TAn

Rotate angle 'n'. 'n' is equivalent to degrees in the range -360 to 360. If 'n' is positive, rotation is counter-clockwise, if 'n' is negative, rotation is clockwise. If 'n' is outside the specified range, an "Illegal function call" error occurs.

Cn

Set color 'n' (from 0 to 3 in medium resolution, and 0 to 1 in high or super resolution).

Sn

Set scale factor. 'n' may range from 1 to 255. The scale factor multiplied by the distances given with U,D,L,R,E,F,G,H or relative M commands gives the actual distance traveled.

Xstringexp

Execute substring. This powerful command allows you to execute a second substring from a string.

Pn.m

'n' is the color chosen to paint the interior of the closed figure and 'm' is the border color. You must specify both parameters or an error will occur. Both parameters can range from 0 to 3 in medium resolution and from 0 to 1 in high or super resolution mode.

Remarks

In all GML commands, "n," "x," and "y" arguments can be constants like "327" or "=numvar;." The semicolon is necessary if you enter a variable this way or if you use the X command; otherwise you can omit the semicolon between commands. Spaces are ignored in "stringexp." For example:

M + = A;, - = B;

Examples

To draw a box:

10 SCREEN 1 20 A = 40

30 DRAW "U=A; R=A; D=A; L=A;"

10 U\$ = "U30;" : D\$ = "D30;"

15 L\$ = "L40;" : R\$ = "R40;"

20 BOX\$ = U\$ + R\$ + D\$ + L\$

30 DRAW "XBOX\$;"

40 REM DRAW "XUS;XRS;XDS;XLS;"

50 'would have drawn the same box

EDIT Command

Lets you change a program line. EDIT is only used in immediate mode.

Syntax

EDIT [linenum].]

linenum

is the program line number. If no such line exists, an "Undefined Line number" error message is displayed.

A period can be used instead of a line number to refer to the current line.

Remarks

When you enter an EDIT command, GWBASIC displays the specified line and positions the cursor under the first digit of the line number. The line may then be modified by using the special editor keys.

The EDIT command can be used to redisplay and edit a line which has just been entered. (The line number symbol "." always refers to the current line.)

LIST may also be used to display program lines for editing.

Example

EDIT 500 500 PRINT AS,B\$,C\$ Terminates program execution, closes all open data files, and returns to command level. END is only used in a program.

Syntax

END

Remarks

END statements may be placed anywhere in the program to terminate execution. Unlike the STOP statement, END does not cause a "Break in line nnnnn" message to be printed. An END statement at the end of a program is optional. GWBASIC always returns to command level after an END is executed.

Example

520 IF K>1000 THEN END ELSE GO TO 20 Allows modification of parameters in GWBASIC's Environment String Table.

Syntax

ENVIRON parm-id = [text[;]]

parm-id

is a valid string expression containing the new Environment String parameter

Remarks

The ENVIRON statement may be used, for example, to change the "PATH" parameter for a child process. Parameters may also be passed to a child process by inventing a new environment parameter.

Rules

- "parm-id" is the name of the parameter such as "PATH."
- "parm-id" must be separated from 'text' by '=' or ' '(blank) such as "PATH='. ENVIRON takes everything to the left of the first blank or "=" as the "parm-id," and everything to the right as 'text.'
- "text" is the new parameter text. If "text" is a null string, or consists only of ";" (a single semicolon, such as "PATH=;") then the parameter (including "parm-id=") is removed from the Environment String Table and the Table is compressed.

- If "parm-id" does not exist in the Environment String Table, then "parm-id" is added at the end of the Environment String Table.
- If "parm-id" does exist, it is deleted, the Environment String Table is compressed, and the new "parm-id" is added at the end.

Examples

The following MS-DOS command will create a default "PATH" to the Root Directory on Disk A:

PATH = A:

The PATH may be changed to a new value by:

ENVIRON"PATH = A:SALES; A:ACCOUNTING"

A new parameter may be added to the Environment String Table:

ENVIRON "SESAME = PLAN"

The Environment String Table now contains:

PATH = A:SALES;A:ACCOUNTING SESAME = PLAN

If you then entered:

ENVIRON "SESAME = ;"

then you would have deleted SESAME, and you would have a table containing:

PATH = A:SALES;A:ACCOUNTING

ENVIRON Statement

Possible errors

- "Type mismatch" if "parm" is not a string.
- "Out of Memory" if the Environment Table is full and no more can be allocated.

Allows you to retrieve the specified Environment String from GWBASIC's Environment String Table.

Syntax

ENVIRON\$ [(parm) | (nth_parm)]

parm

nthparm

is a string expression containing the parameter to be retrieved is an integer expression returning a value in the range 1 to 255

Remarks

- If a string argument is used, ENVIRON\$ returns a string containing the text following "parm=" from the Environment String Table.
- If "parm=" is not found, or no text follows "parm=" then a null string is returned.
- If a numeric argument is used, ENVIRON\$
 returns a string containing the "nth_parm"
 from the Environment String Table including
 the "parm"=text".
- If there is no "nth_parm" then a null string is returned.

Possible errors

- "Illegal function call"—If"nth_parm" is out of range.
- "Type Mismatch"—If "parm" is not a string. "String too long" If the string is longer than 255 characters

EOFFunction

Indicates that the end of a file has been reached.

Syntax

EOF (filenum)

filenum

is the file number specified in the $\ensuremath{\mathsf{OPEN}}$ statement

Remarks

For sequential files, the EOF function returns true (-1) if there is no more data in the file and false (0) if end-of-file has not been reached. Use EOF to test for end-of-file while inputting, to avoid "Input past end" errors.

EOF is significant only for a file opened for sequential input from disk, or for a communications file. A true value (-1) for a communications file means that the buffer is empty.

EOF (0) returns the end of file condition on standard input devices used with redirection of I/O.

Example

5 DIM M(500) 10 OPEN "I",1,"DATA" 20 K = 0 30 IF EOF(1) THEN 100 40 INPUT #1,M(K) 50 K = K + 1:GOTO 30 100 REM PROCESS DATA

This example reads data from the sequential fine named "DATA." Values are read into array M until the end of file is reached.

Releases space and variable names previously reserved for arrays. The data is lost and the array(s) no longer exist.

Syntax

ERASE array [, array]...

array

is the name of an array to be erased.

Remarks

Arrays may be redimensioned after they are ERASEd, or the previously allocated array space in memory may be used for other purposes. If an attempt is made to redimension an array without first ERASEing it, a "Duplicate Definition" error occurs.

Example

It is not good programming practice to reuse an identifier. This may generate errors or reduce the program readability. You may, however, find it useful to redeclare an erased array; for example, when an array name is known by a subroutine and you want to pass arrays with different number of dimensions of subscript upper bounds to this subroutine.

10 DIM A(15,15),B(10,20) 100 ERASE A,B 110 DIM A (100),B(2,2,2)

Upon execution of statement 100, arrays A and B are deleted and the corresponding memory space is made free. You may define other arrays (see statement 110) with the same names but different numbers of dimensions and upper bounds.

ERDEV is an integer function which contains the error code returned by the last device to declare an error.

ERDEV\$ is a string function which contains the name of the device driver which generated the error.

Syntax

[ERDEV | ERDEV\$]

ERDEV is set by the Interrupt X'24' handler, when an error within MS-DOS is detected. ERDEV will contain the INT 24 error code in the lower 8 bits, and the upper 8 bits will contain the "Word attribute bits" (b15-b13) from the Device header block. If the error was on a character device, ERDEV\$ will contain the 8-byte character device name.

If the error was not on a character device, ERDEV\$ will contain the two character block device name (A:, B:, C: etc).

Example

User installed device driver "MYLPT2" caused a "Printer out of Paper" error via INT 24.

If the driver's error number for that problem was 9, ERDEV contains the error number 9 in the lower 8 bits and the device header word attributes in the upper 8 bits.

ERDEV\$ contains "MYLPT2."

The ERR function returns the error code and the ERL function returns the number of the line which contains the error.

Syntax

[ERR | ERL]

Remarks

When an error handling routine is entered, the function ERR contains the error code and the function ERL contains the line number of the line in which the error was detected.

The ERR and ERL functions are usually used in IF...THEN statements to direct program flow in the error handling routine.

If the statement that caused the error was a direct mode statement, ERL will contain 65535.

If the line number is not on the right side of the relational operator, it cannot be renumbered with RENUM. Because ERL and ERR are reserved functions, neither may appear to the left of the equal sign in a LET (assignment) statement.

GWBASIC error codes are listed in Appendix A.

To test whether an error occurred in a direct statement, use IF 65535=ERL THEN __

Otherwise, use

IF ERR=error code THEN_ IF ERL=line number THEN_

Example

LIST
10 REM RECTANGLE2
20 ON ERROR GOTO 70
30 INPUT "Length and Width";L,W
40 IF (L<0) OR (W<0) THEN ERROR 200
50 PRINT "Area=';L*W;" L=';L;" W=';W
60 GOTO 30
70 IF (ERR=200) AND (ERL=40)
THEN PRINT "L or W<0':RESUME 30

80 ON ERROR GOTO 0
90 END
Ok
RUN
Length and Width?
-2,5
L or W<0
Length and Width?
2,5
Area = 10 L = 2 W = 5
Length and Width?
C
Break in 30
Ok

If you enter a negative value for L or W, the error handling routine is activated and the system displays:

L or W<0

Execution is resumed at statement 30 (see RESUME statement below). Note the use of ERR and ERL functions in the error handling routine.

Simulates the occurrence of a GWBASIC error, or generates a user defined error.

Syntax ERROR n

n is an integer expression representing an error code. It must be greater than 0 and less than or equal to 255. If it is not an integer,

it is rounded to the nearest integer.

Remarks If the value of the integer expression equals an

error code already in use by GWBASIC, then

the ERROR is simulated, and the

corresponding error message will be displayed.

Example LIST

10 S = 10

20 T=5

30 ERROR S+T

40 END

Ok

RUN

String too long in line 30

Ok

Or, in immediate mode:

ERROR 15 String too long If the value of the numeric expressions is greater than any error codes used by GWBASIC, then the ERROR statement will generate a user-defined error. This user-defined error code may then be handled in the error trapping routine (see the ON ERROR statement in this chapter).

Note: To define your own error, use a value that is greater than any used by GWBASIC error codes. (It is preferable to use the highest available values, so compatibility may be maintained if more error codes are added to GWBASIC.)

If an error statement specifies a code for which no error message has been defined, then GWBASIC responds with the message: Unprintable error. Returns "e" (base of natural logarithms) to the power of the argument.

Syntax

EXP(numexp)

Remarks

"numexp" must be <=87.3365. If EXP overflows, the "Overflow" error message is displayed, machine infinity with the appropriate sign is supplied as the result, and execution continues.

EXP is calculated in single precision, unless "/D" is supplied in the GWBASIC command line

Example

10 X = 5 20 PRINT EXP(X-1) RUN 54.59815 Ok Allocates space for variables in a random file buffer

FIELD is always used in a program.

Syntax

FIELD [#]filenum,width AS stringvar [,width AS stringvar]...

filenum

is the number under which the file was OPENed

width

is the number of characters to be allocated to stringvar

stringvar

is a string variable name that will be used for random file access

Remarks

A FIELD statement must be executed to format the random file buffer, before a GET statement or PUT statement can be executed.

The total number of bytes allocated in a FIELD statement cannot exceed the record length that was specified when the file was OPENed. Otherwise, a "Field overflow" error occurs. (The default record length is 128 bytes.)

Any number of FIELD statements may be executed for the same file. All FIELD statements that have been executed remain in effect at the same time. Each new FIELD statement redefines the buffer from the first character position. There may be multiple FIELD definitions for the same data.

Do not use a FIELDed variable name in an input statement or to the left of the equal sign in an assignment statement. Once a variable name is FIELDed, it points to the correct place in the random file buffer. If a subsequent INPUT or LET statement with that variable name on the left side of the equal sign is executed, the variable no longer refers to the random file buffer, but to the variables stored in string space.

If previously defined in a FIELD statement, a variable name may be inserted to the right of the equal sign in an assignment statement.

Example 1

10 FIELD 1,20 AS N\$,10 AS ID\$,40 AS ADD\$

Allocates the first 20 positions (bytes) in the random file buffer to the string variable N\$, the next 10 positions to ID\$, and the next 40 positions to ADD\$. FIELD does NOT place any data in the random file buffer. (See also "GET" and "LSET/RSET" in this chapter.)

Example 2

10 OPEN "R",#1,"A:PHONELST",35
15 FIELD #1,2 AS RECNBR\$,33 AS DUMMY\$
20 FIELD #1,25 AS NAME\$,10 AS PHONENBR\$
25 GET #1
30 TOTAL = CVI(RECNBR)\$
35 FOR I = 2 TO TOTAL
40 GET #1, I
45 PRINT NAME\$, PHONENBR\$
50 NEXT I

Illustrates a record with multiply defined fields. In statement 15, the 35 byte field is defined for the first record to keep track of the number of records in the file. In the next loop of statements (35-50), statement 20 defines the field for individual names and phone numbers.

Example 3

10 FOR LOOP% = 0 TO 7
20 FIELD #1,(LOOP%* 16) AS OFFSETS,16 AS AS(LOOP%)
30 NEXT LOOP%

Shows the construction of a FIELD statement using an array of elements of equal size. The result is equivalent to the single declaration:

FIELD #1,16 AS AS(0),16 AS AS(1),..., 16 AS AS(6),16 AS AS(7)

Example 4

- 10 DIM SIZE%(4%): REM ARRAY OF FIELD SIZES
- 20 FOR LOOP% = 0 TO 4%:READ SIZE% (LOOP%): NEXT LOOP%
- 30 DATA 9,10,12,21,41
- 120 DIM A\$(4%): REM ARRAY OF FIELDED VARIABLES
- 130 OFFSET% = 0
- 140 FOR LOOP% = 0 TO 4%
- 150 FIELD #1,0FFSET%AS OFFSET\$,SIZE%(LOOP%)AS A\$(LOOP%)
- 160 OFFSET% = OFFSET% + SIZE%(LOOP%)
- 170 NEXT LOOP%

Creates a field in the same manner as Example 3. However, the element size varies with each element. The equivalent declaration is:

FIELD #1,SIZE%(0) AS A\$(0),SIZE%(1) ASA\$(1),_ SIZE%(4%) AS A\$(4%)

Example 5

10 FIELD#1.225 AS TSTS

Make sure to observe the maximum length restriction for various variables. For example, in the FIELD statement above the maximum length of TST\$ is 255.

Displays the names of files in the specified directory.

Syntax

FILES [filename]

filename

is a string expression including either a filename or a pathname and optional device designation.

Remarks

If "filename" is omitted, all the files on the currently selected drive will be listed. "filename" is a string formula which may contain question marks (?) or asterisks (*) used as wild cards. A question mark will match any single character in the filename or extension. An asterisk will match one or more characters starting at the position. The asterisk is a shorthand notation for a series of question marks. The asterisk need not be used when all the files on a drive are requested, e.g., FILES "B:".

If a filename is used and no explicit path is given, the current directory is the default.

Examples

FILES

Show all files on the current directory

FILES "*.BAS"

Shows all files with an extension of .BAS

FILES "A:*.*"

Shows all files on drive A

FILES "A:"

Equivalent to the preceding example

FILES "GEO?.BAS"

Shows all files on the current directory of the MS-DOS default drive that have a filename of 4 characters beginning with GEO and an extension of .BAS

Sub-directories are denoted by <DIR> following the directory name.

FILES "SALES\"

Lists the files in the subdirectory SALES.

FILES "SALES*.BAS"

Lists the files in the subdirectory SALES that have the extension .BAS.

FIX Function

Returns the truncated integer part of the

argument.

Syntax

FIX(numexp)

Remarks

FIX(numexp) is equivalent to

SGN(numexp)*INT(ABS(numexp)). The major difference between FIX and INT is that FIX does not return the next lower number for a

negative argument.

Examples

PRINT FIX(58.75)

58 Ok

PRINT FIX(-58.75)

-58 Ok

FOR...NEXT Statements

Allow a series of statements to be performed in a loop a specified number of times.

Syntax FOR numvar=x TO y [STEP Z]

.

NEXT [numvar] [,numvar]

numvar is an integer or single-precision variable used as a counter
 x is a numeric expression representing the initial counter value
 y is a numeric expression representing the final counter value
 Z is a numeric expression used as an increment

Remarks

The program lines following the FOR statement are executed until the NEXT statement is encountered. Then the counter (numvar) is incremented by the amount specified by STEP (Z). A check is performed to see if the value of the counter is now greater than the final value (Y). If it is not greater, GWBASIC branches back to the statement after the FOR statement and the process is repeated. If it is greater, execution continues with the statement following the NEXT statement. This is a FOR...NEXT loop.

If STEP is not specified, the increment is assumed to be one. If STEP is negative, the final value of the counter is set to be less than the initial value. The counter is decreased each time through the loop. The loop is executed until the counter is less than the final value.

The counter must be an integer or single precision numeric constant. If a double precision numeric constant is used, a "Type mismatch" error results.

The body of the loop is skipped if the initial value of the loop times the sign of the STEP exceeds the final value times the sign of the STEP.

Nested Loops

FOR...NEXT loops may be nested. A nested loop may be placed within the context of another FOR...NEXT loop. When loops are nested, each loop must have a unique variable name as its counter. The NEXT statement for the inside loop must appear before that for the outside loop. If nested loops have the same end point, a single NEXT statement may be used for all of them. A statement of this form:

NEXT V1, V2, V3

performs the same action as this sequence of statements:

NEXT V1 NEXT V2 NEXT V3

The variable(s) in the NEXT statement may be omitted, in which case the NEXT statement matches the most recent FOR statement.

If a NEXT statement is encountered before its corresponding FOR statement, a "NEXT without FOR" error message is issued and execution is terminated.

```
Example 1
             10 K=10
             20 FOR I = 1 TO K STEP 2
             30 PRINT I;
             40 K=K+10
             50 PRINT K
             60 NEXT
             RUN
               1 20
               3 30
               5 40
               7 50
               9 60
             Ok
Example 2
              10 J=0
             20 FOR I=1 TO J
             30 PRINT I
             40 NEXT I
```

In this example, the loop does not execute because the initial value of the loop exceeds the final value.

In this example, the loop executes ten times. The final value for the loop variable is always set before the initial value is set. Returns the number of bytes in memory not being used by GWBASIC.

Syntax

FRE(dummy)

Remarks

Strings in GWBASIC have variable lengths, changing each time you assign a value. This dynamic manipulation of strings can cause the string space in memory to become fragmented.

The free function can reorganize the string space in memory. This housekeeping consolidates the free space.

The argument to FRE is a dummy argument. Be patient: housekeeping may take 1 to 1-1/2 minutes.

GWBASIC, itself, initiates housekeeping when free memory is used up. If you are doing extensive manipulation of variable length strings use FRE(" ") periodically.

Example

PRINT FRE(0) 14542 Ok

GET (COM files)

Statement

Reads a specified number of bytes into the communications buffer.

Syntax GET [#]filenum, length

filenum is an integer expression returning a valid file number

length is an integer expression returning the number of bytes to be transferred into the communications buffer. Length cannot exceed the value set by the /S: switch when GWBASIC was invoked, or the value optionally given, in the OPEN statement for the device.

Reads a record from a random disk file into a random buffer.

Syntax GET [#]filenum[,recordnum]

filenum is the number under which the file was OPENed

recordnum is the number of the record to be read, in the range 1 to 32,767. If it is omitted, the next record (after the last

GET) is read into the buffer.

Remarks The largest possible record number is 32,767.

After a GET statement is executed, INPUT# or LINE INPUT# are executed to read characters from the random file buffer. If a FIELD statement has been executed, the characters can be accessed through the variable defined in the FIELD statement.

Example

```
10 OPEN "r",1,"A:RAND",48
20 FIELD# 1,20 AS R1$,20 AS R2$,8 AS R3$
30 FOR L = 1 TO 2
40 GET# 1,L
50 PRINT R1$,R2$,CVD(R3$)
60 NEXT
70 CLOSE# 1
80 END
0k
RUN
Super man USA 11234621
robin hood England 23462101
0k
```

This program retrieves information stored in the specified file. The data read into the buffer may be accessed by the program. This is done here by the PRINT statement at line 50. These data items were written to the file by the PUT-File statement. Reads graphic images from the screen.

Syntax [GET] (x1,y1)-(x2,y2), array

(x1,y1)-(x2,y2) are coordinates in either absolute or relative form defining a

screen area

array is the name assigned to the array that will hold the image

Remarks

The GET statement should be used in conjunction with the PUT statement. GET transfers the screen image bounded by the rectangle described by the specified points into the array. The rectangle is defined the same way as the rectangle drawn by the LINE

statement using the ",B" option.

PUT transfers graphics images to the screen. GET and PUT permit animation and high-

speed object motion.

The array must be numeric, but may be any precision.

7-117

Array Dimensions

The storage format in the array is as follows:

- 2 bytes giving x dimension in BITS
- 2 bytes giving y dimension in BITS

The data for each row of pixels is left justified on byte boundaries. If the screen image is not an even multiple of 8 bits, zero padding occurs to the byte boundary. The required array size in bytes is:

4 + INT((x*bitsperpixel + 7)/8)*y

"bitsperpixel" is 2 for medium resolution, and 1 for high and super resolution.

The bytes per element of an array are:

- 2 for integer
- 4 for single precision
- 8 for double precision

Example

If you want to GET a 10 by 12 image into an integer array, the number of bytes required is 4+INT((10*2+7)/8)*12 or 40 bytes. You need an integer array with at least 20 elements.

It is possible to examine the "x" and "y" dimensions and even the data itself if an integer array is used. The "x" dimension is in element 0 of the array, and the "y" dimension is found in element 1. Integers are stored low byte first, then high byte, but the data is transferred high byte first (leftmost) and then low byte.

GOSUB transfers control to a GWBASIC subroutine by branching to the specified line. RETURN transfers control to the statement following the most recent GOSUB (or ON...GOSUB) executed, or to a specified line.

Syntax GOSUB [linenum1] RETURN [linenum2]

linenum1 is the first line number of the subroutine

is any line of your program different from linenum1 and from the line number of the GOSUB statement linenum2

Remarks

A subroutine may be called any number of times in a program. A subroutine may also be called from within another subroutine. Such nesting of subroutines is limited only by available memory.

The RETURN statement(s) in a subroutine causes GWBASIC to branch back to the statement following the most recent GOSUB or ON...GOSUB statement executed. A subroutine may contain more than one RETURN statement, if logic dictates a return at different points in the subroutine.

The "linenum2" option may be included in the RETURN statement to return to a specific line number from the subroutine. Use this type of return with care, however, because any other GOSUBs, WHILEs, or FORs that were active at the time of the GOSUB will remain active, and errors such as "FOR without NEXT" may result.

Subroutines may appear anywhere in the program, but it is recommended that the subroutine be readily distinguishable from the main program. To prevent inadvertent entry into the subroutine, precede it with a STOP, END, or GOTO statement that directs program control around the subroutine.

If either "linenum1" or "linenum2" does not exist in the program, an "Undefined line number" error is returned.

Example

10 GOSUB 40

20 PRINT "BACK FROM SUBROUTINE"

30 END

40 PRINT "SUBROUTINE";

50 PRINT "IN";

60 PRINT "PROGRESS"

70 RETURN

RUN

SUBROUTINE IN PROGRESS BACK FROM SUBROUTINE

Ok

Transfers control to a specified program line.

Syntax GOTO linenum

linenum is the number of a line in the program

Remarks

If linenum is the line number of an executable statement, that statement and those following are executed. If it is the line number of a nonexecutable statement, execution proceeds at the first executable statement encountered after linenum. If the specified "linenum" does not exist in the program, an "Undefined line number" error is returned.

Example

10 READ R
20 PRINT "R = ';R,
30 A = 3.14*R \u03b3 2
40 PRINT "AREA = ';A
50 GOTO 10
60 DATA 5,7,12
Ok
RUN
R = 5 AREA = 78.5
R = 75 AREA = 153.86
R = 12 AREA = 452.16
?Out of data in 10
Ok

Initializes GWBASIC and the operating environment (GWBASIC is an MS-DOS command, not a GWBASIC command).

Syntax

[GWBASIC [filename] [<stdin] [>stdout] [/F:number of files] [/S:lrecl] [/C:buffer size] [/M:[highest memory] [,max block size]] [/D] [/I]

Options beginning with a slash (/) are called switches. A "switch" is a means used to specify parameters.

filename

is a string literal (not included in quotation marks) that specifies a GWBASIC program file. If the file is present, GWBASIC proceeds as if a RUN 'filename' command were given after initialization is complete.

A default extension of .BAS is used if none is supplied and the filename is less than 9 characters long. The 'filename' option allows GWBASIC programs to be run in batch by putting this form of the command line in an AUTO-EXEC.BAT file. GWBASIC programs which run this way will need to exit via the SYSTEM command in order to allow the next command from the AUTOEXEC.BAT file to be executed.

stdin

is a literal string (not included in quotation marks) for the standard input file specification. GWBASIC input is redirected from the file specified by 'stdin'. When present, this syntax must appear before any switches. (See "Re-direction of Standard Input and Output below.)

stdout

is a literal string (not included in quotation marks) for the standard output file specification. GWBASIC is redirected to the file specified by 'stdout.' When present, this syntax must appear before any switches. (See "Re-direction of Standard Output" below.)

/F:

this switch sets the maximum number of files that may be open simultaneously during the execution of a GWBASIC program. It is ignored unless the /I switch is specified on the command line. Refer to the /I switch below.

If this switch and the /I switch are present, then the maximum number of files is set to 'files'. Each file requires 62 bytes for the File Control Block (FCB) plus 128 bytes for the data buffer. The data buffer size may be altered via the /S: option switch. If the /F option is omitted, the number of files is set to 3.

The number of open files that MS-DOS supports depends upon the value of the FILES = parameter in the CON-FIG.SYS file. It is recommended that FILES = 10 for GWBASIC. Remember that the first 3 are taken by 'stdin', 'stdout', 'stderr', 'stdaux', and 'stdprn'. One additional file handler is needed by GWBASIC for LOAD, SAVE, CHAIN, NAME and MERGE. This leaves 6 for GWBASIC File I/O, thus /F:6 is the maximum supported by MS-DOS when FILES=10 appears in the CONFIG.SYS file. Attempting to OPEN a file after all the file handlers have been exhausted will result in a "Too many files" error.

GWBASIC Command

IS:

this switch sets the maximum record length allowed with random files. It is ignored unless the /I switch is specified on the command line (refer to the /I switch below). If this switch and the /I switch are present, then the maximum record length is set to 'lrecl'. The record length option ('recordlength') on the OPEN statement cannot exceed this value. If the /S: option is omitted, the record length defaults to 128 bytes. The maximum value permitted for 'lrecl' is 32767 bytes.

IC:

buffersize if present, controls RS232 Communications. If RS232 cards are present, /C:0 disables RS232 support. Any subsequent I/O attempts will result in a "Device unavailable" error. Specifying /C:n allocates 'n' bytes for the receive buffer for each RS232 card present. If the /C: option is omitted, GWBASIC allocates 256 bytes for the receive buffer of each card present. GWBASIC ignores the /C: switch when RS232 cards are not present.

/M:[highest memory][,maxblock size]

when present, 'highest memory' sets the maximum number of bytes that will be used as GWBASIC workspace. GWBASIC will attempt to allocate 64K of memory for the data and stack segment. If machine language subroutines are to be used with GWBASIC programs use the /M: switch to set the highest memory location that GWBASIC can use. When omitted pr 0. GW BASOC attempts to allocate all it can up to a maximum of 65536 bytes.

If order to load programs above the GWBASIC workspace you must use the optional parameter 'max blocksize' to reserve areas for the workspace and your programs. 'Maxblocksize' must be in Paragraphs (byte multiples of 16). When omitted, &H1000 (4096) is assumed. This allocates 65536 bytes (65536=4096 x 16) for GWBASIC's Data and Stack segment. If you require 65536 bytes for GWBASIC and 512 bytes for machine language subroutines, then use /M:,&H1010 (4096 paragraphs for GWBASIC + 16 paragraphs for your routines). /M:,2048 says: "Allocate and use 32768 bytes maximum for data and stack". /M:32000,2048 allocates 32768 bytes maximum but GWBASIC will only use the lower 32000. This leaves 768 bytes available for program space.

 $/\mathbf{D}$

if present, causes the Double Precision Transcendental maths package to remain resident. The functions that will be calculated in double precision if this package is resident are: ATN, COS, EXP, LOG, SIN, SQR, and TAN. If omitted, this package is discarded and the space is freed for program use. The amount of memory required by this package is approximately 3,000 bytes.

/I

GWBASIC is able to dynamically allocate space required to support file operations. For this reason GWBASIC does not need to support the /S and /F switches. However, some applications are written in such a manner that certain BASIC internal data structures must be static. In order to provide compatibility with these BASIC programs, GWBASIC will statically allocate space required for file operations based on the /S and /F switches when the /I switch is specified.

Note: "number of files," "lrecl," "buffer size," "highest memory" and "max block size" are numbers that may be Decimal, Octal (preceded by &0), or Hexadecimal (preceded by &H).

Examples

A>GWBASIC PAYROLL

Uses 64k of memory and 3 files, loads and executes PAYROLL BAS.

A>GWBASIC INVENT/F:6

Uses 64k of memory and 6 files, loads and executes INVENT.BAS.

A>GWBASIC /C:0/M:32768

Disables RS232 support and uses only the first 32k of memory.

A > GWBASIC /F:4/S:512

Uses 4 files and allows a maximum record length of 512 bytes.

A>GWBASIC TTY/C:512

Uses 64k of memory and 3 files, allocates 512 bytes to RS232 receive buffers, load and execute TTY.BAS.

Redirection of Standard Input and Output

Under GWBASIC you can redirect your Input and Output. Generally, standard input is read from the keyboard, but this can be redirected to any file specified on the GWBASIC command line. Standard output, generally written to the screen, can be redirected to any device or file specified on the GWBASIC command line.

- When redirected, all INPUT, LINE INPUT, INPUT\$ and INKEY\$ statements read from the "stdin" specified instead of from the keyboard.
- All PRINT statements write to the "stdout" specified instead of the screen.
- Error messages go to standard output.
- File input to "KYBD:" reads from the keyboard.
- File output to "SCRN:" outputs to the screen.
- GWBASIC continues to trap keys from the keyboard when the ON KEY(n) statement is used.
- The printer echo key does not cause LPT1: echoing if Standard Output has been re-directed.
- Typing CTRL BREAK causes GWBASIC to close any open files, issue the message "Break in line line number>" to standard output, exit GWBASIC, and return to MS-DOS.

• When input is re-directed, GWBASIC continues to read from this source until a CTRL Z is detected. This condition may be tested with the EOF function. If the file is not terminated by a CTRL Z or if a GWBASIC file input statement tries to read past end-of-file, then any open files are closed. The message "Read past end" is then written to standard output, and GWBASIC returns to MS-DOS.

Examples

GWBASIC MYPROG > DATA.OUT

Data read by INPUT and LINE INPUT continue to come from the keyboard. Data output by PRINT goes into the file DATA.OUT.

GWBASIC MYPROG < DATA.IN

Data read by INPUT and LINE INPUT comes from DATA.IN. Data output by PRINT goes to the screen.

GWBASIC MYPROG < MYINPUT.DAT > MYOUTPUT.DAT

Data read by INPUT and LINE INPUT comes from the file MYINPUT.DAT and data output by PRINT goes into MYOUTPUT.DAT.

GWBASIC MYPROG <\SALES JOHN TRANS. \SALES\SALES.DAT

Data read by INPUT and LINE INPUT will now come from the file.

HEX\$

Returns a string which represents the hexadecimal value of the decimal argument.

Syntax

HEX\$(numexp)

Ok

Remarks

"numexp" is rounded to an integer before HEX\$ is evaluated. If "numexp" is negative, the two's complement form is used.

Example

10 INPUT X
20 A\$=HEX\$(x)
30 PRINT X "DECIMAL IS" A\$ " HEXADECIMAL"
RUN
? 32
32 DECIMAL IS 20 HEXADECIMAL

See the OCT\$ function later in this chapter for details on octal conversion.

IF ... GOTO ... ELSE IF ... THEN ... ELSE

Statements

Makes a decision regarding program flow based on the result of a specified condition. IF ... GOTO ... ELSE and IF ... THEN ... ELSE are usually used in a program.

Syntax 1 IF condition GOTO linenum [ELSE (statements | linenum)]

Syntax 2 IF condition [,] THEN (statements|linenum)
[ELSE (statements|linenum)]

condition may be a numeric, relational, or logical expression. GWBASIC determines whether the condition is true or false by testing the result of the expression for non zero and zero respectively. A non zero result is true and a zero result is false. Because of this, you can test whether the value of a variable is non zero or zero by merely specifying the name of the variable as 'condition'.

statements are one or more statements. Each statement must be separated from the preceding one by a colon (:).

linenum is a line number of the program in memory

Remarks

If the result of "condition" is true (not zero), the GOTO or THEN clause is executed. GOTO is always followed by a line number. THEN may be followed by either a line number for branching or one or more statements to be executed. If the result of "condition" is false (zero), the GOTO or THEN clause is ignored and the ELSE clause, if present, is executed. Execution continues with the next executable statement.

Nesting of IF Statements

IF... THEN ... ELSE statements may be nested. Nesting is limited only by the length of the line. For example:

IF X>Y THEN PRINT "GREATER" ELSE IF Y>X THEN PRINT "LESS THAN" ELSE PRINT "EQUAL"

is a legal statement. If the statement does not contain the same number of ELSE and THEN clauses, each ELSE is matched with the closest unmatched THEN. For example:

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

will not print "A < >C" when A < >B.

If an IF...THEN statement is followed by a line number in direct mode, an "Undefined line" error results, unless a statement with the specified line number had previously been entered in indirect mode.

Note: When using IF to test equality for a value that is the result of a floating-point computation, remember that the internal representation of the value may not be exact. Therefore, the test should be against the range over which the accuracy of the value may vary. For example, to test a computed variable A against the value 1.0, use:

IF ABS (A-1.0) < 1.0E-6 THEN ...

This test returns true if the value of A is 1.0 with a relative error of less than 1.0E-6.

Example 1 200 IF I THEN GET#1,I

This statement GETs record number I if I is not zero.

Example 2 100 IF(I<20)*(I>10) THEN DB=1979-1:GOTO 300 110 PRINT "OUT OF RANGE"

In this example, a test determines if I is greater than 10 and less than 20. If I is in this range, DB is calculated and execution branches to line 300. If I is not in this range, execution continues with line 110.

Example 3 210 IF IOFLAG THEN PRINT AS ELSE LPRINT AS

This statement causes printed output to go either to the terminal or the line printer, depending on the value of the variable IOFLAG. If IOFLAG is zero, output goes to the line printer; otherwise, output goes to the screen.

Returns either a one or two character string read from the keyboard. INKEY\$ is always used in a program.

Syntax

INKEYS

Remarks

INKEY\$ returns one of the following values:

- a null string if no character is read from the keyboard
- a one-character string in accordance with a single character read from the keyboard
- a two-character string in accordance with an extended ASCII code. The first character is zero; the second indicates the scan code of the key pressed (refer to Appendix C; Extended Codes).

Although more than one character may be pending in the keyboard buffer, a single character only will be read. This value must then be assigned to a variable before it is used by the GWBASIC program.

The following control characters can be entered at the keyboard, and will not be passed to the program:

- PRTSC print the screen
- CTRL NUMLOCK set the system to pause
- CTRL BREAK stop the program
- ALT CTRL DEL reset the system

Note that CR is passed to the program like any character string.

Example

1000 'Timed input Subroutine
1010 RESPONSES = ''''
1020 FOR 1% = 1 TO TIMELIMIT%
1030 AS = INKEYS:IF LEN(AS) = 0 THEN 1060
1040 IF ASC(AS) = 13 THEN TIMEOUT% = 0:RETURN
1050 RESPONSES = RESPONSES + AS
1060 NEXT 1%
1070 TIMEOUT% = 1:RETURN

This subroutine returns two values:

- RESPONSE\$ which contains the string entered from keyboard
- TIMEOUT% which equals 0 if the user enters a string of characters from keyboard before a specified number of loops (TIMELIMIT%); otherwise equals 1.

Returns the byte read from a port.

Syntax INP(port)

port is a valid port number in the range 0 through 65535

Remarks INP is the complementary function to the

OUT statement. Allows input from the keyboard during program execution. INP

is only used in a program.



Allows input from terminal during program execution.

Syntax

INPUT[;][prompt\$;]variable[\$,variable]...

prompt

is a string constant enclosed in quotation marks which prompts for the necessary input

variable

is a numeric or string variable which receives the input

Remarks

When an INPUT statement is encountered, program execution pauses and a question mark is printed to indicate the program is waiting for data. If "PROMPT" is included, the string is printed before the question mark. The required data is then entered at the terminal.

A comma may be used instead of a semicolon after the prompt string to suppress the question mark.

If INPUT is immediately followed by a semicolon, then the CR typed by the user to input data does not echo a CR LF sequence.

The data that is entered is assigned to the variable(s) given in the variable list. The number of data items supplied must be the same as the number of variables in the list. Data items must be separated by commas.

The variable names in the list may be numeric or string variable names (including subscripted variables). The type of each data item that is input must agree with the type specified by the variable name. Strings input to an INPUT statement need not be surrounded by quotation marks.

Responding to INPUT with too many or too few items or with the wrong type of value (numeric instead of string, etc.) causes the message "?Redo from start" to be printed. No assignment of input values is made until an acceptable response is given.

The user may use all the GWBASIC screen editor features in responding to INPUT and LINE INPUT statements.

Example

10 INPUT X

20 PRINT X "SQUARED IS" X^2

30 END RUN ? 5

5 SQUARED IS 25

Ok

Example

10 PI = 3.14

20 INPUT "WHAT IS THE RADIUS";R

30 A=PI*R^2

40 PRINT "THE AREA OF THE CIRCLE IS";A

50 PRINT 60 GOTO 20

Ok RUN

WHAT IS THE RADIUS? 7.4

THE AREA OF THE CIRCLE IS 171.9464

WHAT IS THE RADIUS?

etc.

Reads data items from a sequential disk file and assigns them to program variables. INPUT# is usually used in a program.

Syntax

INPUT#filenum,variable [,variable]____

filenum

is the number used when the file was OPENed for input

variable

is a numeric or string variable which will receive a data item from the file. (The type of data in the file must match the type specified by the variable name.) With INPUT#, no question mark is printed, as with INPUT.

Remarks

The data items in the file should appear just as they would if data were being typed in response to an INPUT statement. With numeric values, leading spaces, carriage returns, and line feeds are ignored. The first character encountered that is not a space, carriage return, or line feed is assumed to be the start of a number. The number terminates on a space, carriage return, line feed, or comma.

If GWBASIC is scanning the sequential data file for a string item, leading spaces, carriage returns, and line feeds are also ignored. The first character encountered that is not a space, carriage return, or line feed is assumed to be the start of a string item. If this first character is a quotation mark ("), the string item will consist of all characters read between the first quotation mark and the second. Thus, a quoted string may not contain a quotation mark as a character. If the first character of the string is not a quotation mark, the string is an unquoted string, and will terminate on a comma, carriage return, or line feed (or after 255 characters have been read). If end-of-file is reached when a numeric or string item is being INPUT, the item is terminated.

Example

100 INPUT#1,X\$,Y\$,Z\$

This example uses the INPUT# statement to read three strings from a sequential file into the program.

Returns a string of characters read from the standard input device, the keyboard, or from a file.

Syntax

INPUT\$(length[,[#]filenum])

length

is an integer expression specifying the number of characters to be read from the keyboard or a file

filenum

is the file number specifying the file to be read. If you omit 'filenum', the keyboard is read by default.

Remarks

If the keyboard is used for input, no characters will be displayed on the screen. All characters including control characters are passed through except CTRL BREAK which is used to interrupt the execution of the INPUT\$ function.

When reading COM files, the INPUT\$ function is preferred over INPUT# and LINE INPUT# statements, since all ASCII characters may be significant in communications. INPUT# is least desirable because input stops when a comma (,) or CR is encountered and LINE INPUT# terminates when a CR is encountered.

INPUT\$ allows all characters read to be assigned to a string. INPUT\$ will return x characters from the specified file. The following statements then are most efficient for reading a COM file:

Example 1

'LIST THE CONTENTS OF A SEQUENTIAL FILE IN HEXADECIMAL

10 OPEN"!",1,"DATA" 20 IF EOF(1) THEN 50

30 PRINT HEXS(ASC(INPUTS(1,#1)));

40 GOTO 20 50 PRINT 60 END

Example 2

100 PRINT "TYPE P TO PROCEED OR S TO STOP"

110 X\$=INPUT\$(1) 120 IF X\$="P" THEN 500

130 IF X\$="S" THEN 700 ELSE 100

Example 3

10 WHILE NOT EOF(1)

20 AS=INPUTS(LOC(1),#1)

30 ...

40 ... Process data returned in A\$ _

50 ... 60 WEND

The above sequence of statements reads:

".. While there is something in the output queue, return the number of characters in the queue and store them in A\$. If there are more than 255 characters, only 255 will be returned at a time to prevent "String Overflow". Input continues until the input queue is empty.

(EOF(1) = true.)"



Searches for the first occurrence of a given substring in a string, and returns the position at which the match is found.

Syntax INSTR([start,]string,substring)

start is an integer expression in the range 1 to 255, which specifies where the search is to begin. If omitted, 1 is

assumed.

string is a string expression (in particular a string constant or

variable) whose value is the string to be searched

substring is a string expression (in particular a string constant or

variable) whose first occurrence is to be reached for

Special Values

start >LEN(string)	the returned value is 0
start <1 or start >255	an error is returned (Illegal function call)
'string' is null	the returned value is 0
'substring' cannot be found	the returned value is 0
'substring' is null and start is specified	the returned value is 'start'
'substring' is null and start is omitted	the returned value is 1

Example

10 X\$ = "ABCDEB" 20 Y\$ = "B" 30 PRINT INSTR(X\$,Y\$);INSTR(4,X\$,Y\$) RUN 2 6 Ok

Note: The position at which the match is found is always evaluated from the beginning of the original string, even if start is specified.

INT Function

Returns the largest integer that is equal to, or less than the argument.

Syntax

INT(numexp)

Remarks

Refer to the CINT and FIX functions in this chapter, which also return integer values.

Examples

PRINT INT(99.89)

99 Ok

PRINT INT(-12.11)

-13 Ok Sends a "Control Data" string to a character device driver anytime after the driver has been OPENed.

Syntax IOCTL [#] filenum, string

filenum is the file number open to the Device Driver

string is a string expression containing the Control Data

Remarks

IOCTL commands are generally 2 to 3
characters optionally followed by an
alphanumeric argument. An IOCTL command
string may be up to 255 bytes long.

The IOCTL statement works only if:

- The device driver is installed.
- The device driver processes IOCTL strings.
- GWBASIC performs an OPEN on a file on that device. Most standard MS-DOS device drivers do not process IOCTL strings and it is necessary for the programmer to determine whether the specific driver can handle the command.

If you have installed a driver to replace LPT1 and that driver is able to set page length (the number of lines to print on a page before issuing a form feed), then an IOCTL command to set or change the page length is:

PLn

"n" is the new page length.

Also see the IOCTL\$ Function.

Example 1

• Open the new LPT1 driver and set the Page Length to 66 lines:

10 OPEN "LPT1:" FOR OUTPUT AS #1 20 IOCTL #1,"PL66"

Example 2

Open LPT1 with an initial Page Length of 56 lines

10 OPEN "\DEV\LPT1" FOR OUTPUT AS #1 20 IOCTL #1,"PL56'

You can define other IOCTL commands such as PTn (set Print Tabs every "n" spaces).

Possible Errors

"Bad file number" - IOCTL to a driver that is not OPEN.

"Illegal function call" - if device does not support IOCTL.

"Device Fault" - error in control data.

Returns a "Control Data" string from a Character Device Driver that is OPEN.

Syntax

IOCTL\$([#] filenum)

filenum

is the file number open to the device

Remarks

The IOCTL\$ function is used to receive acknowledgment that an IOCTL statement succeeded or failed or to obtain current status information.

IOCTL\$ can also be used to ask a communications device to return the current baud rate, information on the last error, logical line width, etc.

The IOCTL\$ function works only if:

- The device driver is installed.
- The device driver processes IOCTL strings.
- GWBASIC performs an OPEN on a file on that device.

Also see the IOCTL statement.

Example

10 OPEN "\DEV\FOO" AS #1 20 IOCTL #1,"RAW" 'Tell device that data is "RAW" 30 IF IOCTL\$(1) = "0" THEN CLOSE 1

If Character Driver FOO gives a false return from the Raw data mode IOCTL request, close the files and stop processing.

Possible

"Bad file number" - IOCTL to a driver that is Errors not OPEN.

"Illegal function call" - device does not support IOCTL.

KEY

Statement (Function Keys)

The Key statement performs two entirely different classes of functions depending on the syntax you use. Syntax 1 performs one of the following:

- Sets a function key to act as a "Soft Key", that is, to automatically type any sequence of characters
- Enables or disables the function key display from the 25th line of the screen
- Displays the function key values on the screen

Syntax 2 defines special key sequences so that you can trap for them using the Key (N) Statement.

Syntax1

KEY [OFF|ON|LIST|n,stringexp]

n

is the 'key number'. A constant or a numeric expression in the range of 1 to 10.

stringexp

is a string expression assigned to the key. String constants should be enclosed in quotation marks. The 'stringexp' value may be up to 15 characters long. Longer strings are truncated to 15 characters.

Options

KEY OFF erases the Soft Key display from the bottom line, making this line available for your GWBASIC program. You can use LOCATE 25,1 followed by PRINT to display data on the bottom line of the screen. KEY OFF does not disable the function keys.

KEY ON causes the Soft Key values to be displayed on the bottom line of the screen. If the screen width is 80, all ten Soft Keys are displayed. Five Soft Keys are displayed if the width is 40. In either case, only the first 6 characters of each key value are displayed.

If fewer than the total number of function keys are displayed, you can scroll the function key display (increasing the number of the leftmost key displayed by one each time) by pressing <CTRL><T>. ON is the default state.

KEY LIST displays all Soft Key values on the screen, with all 15 characters of each key displayed.

KEY n,stringexp sets function key n. Any one or all of the ten Function Keys may be assigned up to a 15 byte string by KEY n,stringexp. When the key is pressed, the associated string is input to GWBASIC.

The Soft Keys default to the following values:

F1 - LIST < space > F2 - RUN < CR > F3 - LOAD" F4 - SAVE"
F5 - CONT < CR > F6 - ,"LPT1:" < CR > F7 - TRON < CR > F8 - TROFF < CR > F9 - KEY < space > F10-SCREEN 0,0,0 < CR >

Remarks

- If the function key number is not in the range 1 to 10, an "Illegal function call" error is produced.
- The key assignment string may be 1 to 15 characters in length. If the string is longer than 15 characters, the first 15 characters are assigned.
- Assigning a null string (string of length 0) to a Soft Key disables the Function Key as a Soft Key.
- When a Soft Key is assigned, the INKEY\$ function returns one character of the Soft Key string per invocation.

Examples

50 KEY ON

Displays the Soft Keys on the bottom line.

60 KEY OFF

Erases Soft Key display.

70 KEY 1, MENU + CHR\$(13)

Assigns the string "MENU" CR to Soft Key 1. Such assignments might be used for rapid data entry.

80 KEY 2,

Disables Soft Key 2 as a soft key.

The following routine initializes the first 5 Soft Keys:

1 KEY OFF 'Turn off key display during init.

10 DATA KEY1,KEY2,KEY3,KEY4,KEY5
20 FOR I=1 TO 5:READ SOFTKEYS\$(I)
30 KEY I,SOFTKEYS\$(I)
40 NEXT I
50 KEY ON 'now display new softkeys.

KEY

Statement (ControlKeys)

Syntax 2 of the Key statement defines keys 15 to 20 to allow you to trap any Ctrl-Key, Shift-Key, or Super-Shift (<ALT>)-Key. These are often referred to as "user-defined" keys.

Syntax2

KEY n,

CHR\$(shift) + CHR\$(scan_code)

n

is an integer expression in the range 15 to 20

shift

is a numeric value corresponding to the following hex values for the latched keys:

<CAPS LOCK > &H40 (Caps Lock is active) <NUM LOCK > &H20 (Num Lock is active)

<ALT> &H08 (Alt Key is pressed) <CTRL> &H04 (CTRL Key is pressed) <SHIFT> &H01, &H02, &H03

Both the left and right **SHIFT** keys can be used, where values of &H01, &H02, or &H03 (the sum of hex 01 and hex 02) denote a **SHIFT** key.

You can add multiple shift states, such as <CTRL> and <ALT> keys together, by adding the associated shift state values.

scancode

is a decimal number in the range 1 to 83. It represents the scan code (in decimal) of the key to be trapped. See Appendix A for a complete table of scan codes and their associated key positions.

Remarks

Trapped keys are processed in the following order:

- 1 CTRL PRT SC. CTRL PRT SC produces a printed copy of the screen whether or not you trap for it.
- 2 It is not necessary to define F1 to F10 and the cursor direction keys as trap keys; they are predefined as trap keys.
- 3 The user defined keys are examined (15-20).

Any key that is trapped is not passed to GWBASIC, i.e., it does not go into the keyboard buffer. This applies to any key, including CTRL BREAK or CTRL ALT DEL. By trapping for a key, you can prevent GWBASIC users from accidentally interrupting a program or rebooting the system.

Examples

See the ON KEY(n) GOSUB statement.

Enables or disables event trapping of the specified key.

Syntax

KEY(n) [ON|OFF|STOP]

 \mathbf{n}

is an integer expression in the range 1 to 20, indicating the key to be trapped:

1-10	function keys F1 to F10
11	Cursor up
12	Cursor left
13	Cursor right
14	Cursor down
15-20	keys defined by the form:

To Enable or Disable KEY(n) Trapping

 If a KEY'n' ON is executed, KEY'n' trapping is enabled

KEY n. CHR\$(shift) + CHR\$(scancode)

- If a KEY'n' OFF is executed, KEY'n' trapping is disabled
- If a KEY'n' STOP is executed, KEY'n' trapping is suspended. If KEY(n) is pressed, the event is remembered and an immediate trap occurs, when a KEY'n' ON is executed.

Example

10 KEY 4, SCREEN 0,0,0 'assign softkey 4 20 KEY (4) ON 'enables KEY trapping

:

100 ON KEY (4) GOSUB 1000

.

Key 4 pressed

-

1000 REM KEY (4) Trap Routine

KILL

Command

Deletes a disk file.

Syntax

KILL filename

filename

is a string expression which specifies the file to to be deleted. The filename must include the extension if one exists

Remarks

KILL checks to see if the file is open, and if so displays "File already open". KILL, like OPEN, cannot distinguish a file in another directory from one you may have open. You may get an unexpected "File already open" error under these circumstances.

KILL can only be used to delete a file. Use the RMDIR command to remove a directory.

Example

200 KILL "A:DATA1.DAT"
300 KILL "C:DIR1\DIR2\PROG2.BAS"

Dumps the screen text to the line printer.

Syntax

LCOPY [screentype]

screentype

is an integer expression representing the type of screen in use

Remarks

If the parameter "screentype" is not given, or has the value 0, the text screen is printed on the current system printer.

If "screentype" is greater than 0, and an INT 5 handler (for dumping the screen) is installed, INT 5 is executed. Interrupt 5 is a program for printing the screen bitmap on a graphics printer.

If "screentype" is greater than 0, and an INT 5 handler is NOT installed, the message "Illegal function call" is displayed.

Returns a substring extracting the leftmost number of characters as specified by the "length" parameter.

Syntax LEFT\$(string, length)

string is a string expression whose value is the string from which

the substring is to be returned

length is an integer expression from 0 to 255 which specifies the

number of the characters to be returned.

Remarks If "length" is greater than LEN(string), the entire original string is returned. If "length" =

0, the null string (length zero) is returned.

Refer to the MID\$ and RIGHT\$ functions in

this chapter.

Example Ok

10 A\$ = "GWBASIC"

20 B\$ = LEFT\$(A\$,6)

30 PRINT B\$
GW-BAS

Ok

Returns the number of characters in a given string.

Syntax LEN (stringexp)

stringexp is any string expression whose length is to be returned

Remarks Unprintable characters and blanks are counted in the number of characters. If the argument "stringexp" is a null string, LEN

returns zero.

Example 10 X\$ = "PORTLAND, OREGON" 20 PRINT LEN(X\$)

RUN 16 Ok

In the above example, there are 16 characters in the string "PORTLAND, OREGON" because the comma and the blank space are included.

Assigns a value to a variable.

Syntax [LET] variable=expression

variable is a numeric or string variable which receives the value of

the expression.

expression is the expression whose value is assigned to the variable. The type of expression (numeric or string) must match the type

of the variable; if not, a "Type Mismatch" error occurs.

Remarks In numeric assignments the type of the expres-

sion (integer, single precision, or double precision) may be different from the type of the destination variable. In this case GWBASIC converts the expression value to the type of the variable. Rounding or overflow may occur in

this conversion.

The word LET is optional. The equal sign is sufficient when assigning an expression to a variable name.

Examples 110 LET D = 12

120 LET E = 12/2

130 LET F = 12/4

140 LET SUM = D + E + F

150 A(I) = 300

160 A\$(K) = "ABC"

Draws either a line or a rectangle, or a filled rectangle (Graphics Mode only).

Syntax LINE [STEP] [(x1,y1)]-[STEP] (x2,y2)

[,[color] [B, [F] [,style]]

(x1,y1),(x2,y2) represent absolute co-ordinates or relative coordinates if

STEP is included. If (x1,y1) is omitted the last referenced

point is assumed.

color is the color number specifying the color in which the line or

rectangle is to be drawn (in the range 0 to 3). Refer to the COLOR graphics statement for the current screen mode for

details.

B represents a rectangle

F represents a rectangle to be filled (with color)

style is an optional parameter that may be defined by the user to

produce varying line "styles", i.e., varieties of dotted lines.

Remarks

The following example draws a line from the last point referenced to the point specified (x2,y2). Since no color is specified, the default color is the foreground color.

LINE -(x2,y2)

The examples below specify start and end points in absolute coordinates.

LINE (10,10)-(319,199)

'draws a diagonal line down the screen LINE (10,100)-(319,100)

'draws a horizontal line across the screen

You can specify the color in which the line is drawn:

LINE (15,15)-(25,25),3

draws a line in color 3

The "b" parameter is used to draw a rectangle ("box") in the foreground, where the points (x1,y1) and (x2,y2) represent the opposite corners. In the following example, no color number is specified:

LINE (10,10)-(100,100),,B

draws a box in foreground

Color may be included as follows:

LINE (10,10)-(200,200),2,BF

fills box color 2

The B parameter facilitates the drawing of rectangles, which would otherwise require the following lengthy programming format:

LINE (x1,y1)-(x2,y1) LINE (x1,y1)-(x1,y2) LINE (x2,y1)-(x2,y2) LINE (x1,y2)-(x2,y2)

BF fills the interior of the rectangle with the selected color.

Out-of-range coordinates are not visible on the screen. This is called "line clipping".

If the relative form is used for the second coordinate, it is relative to the first coordinate. For example:

LINE(50,50) -STEP(15,-13)

draws a line from (50,50) to (65,37).

LINE supports the additional argument "style." Style is a 16-bit integer mask used when putting pixels on the screen. This is called "Line-Styling".

Each time LINE plots a pixel on the screen, it will use the current circulating bit in "style." If that bit is 0, no pixel is plotted. If the bit is a 1, then a pixel is plotted. After each pixel, the next bit position in "style" is selected.

Since a 0 bit in "style" does not clear out the old contents, the user may wish to draw a background line before a "styled" line in order to force a known background.

"Style" is used for normal lines and boxes, but has no effect on filled boxes.

LINE (0,0)-(160,100),3,,&HFF00

The example above draws a dashed line from the upper left hand corner to the screen center.

LINE INPUT

Statement

Inputs an entire line (up to 254 characters) to a string variable, without the use of delimiters. A LINE INPUT statement is only used in a program.

Syntax

LINE INPUT [;] [prompt;] stringvar

prompt

is a string constant (enclosed in a pair of quotation marks) that is displayed on the screen before input is accepted. A question mark is not displayed unless it is part of the "prompt" string.

stringvar

is the name of a string variable to which the line is to be assigned.

LINE INPUT Statement

Remarks

All input from the end of "prompt" to the CR is assigned to "stringvar". Trailing blanks are ignored. If a linefeed/carriage return is encountered, both characters are echoed, but the carriage return is ignored, the linefeed is put into "stringvar", and data input continues.

If LINE INPUT is immediately followed by a semicolon, then the CR typed by the user to end the input line does not echo a CR LF sequence on the screen.

You may use all the GWBASIC screen editor features in responding to INPUT and LINE INPUT statements.

Example

See LINE INPUT# statement.

Reads an entire line (up to 254 characters), without delimiters, from a sequential disk data file to a string variable.

Syntax

LINE INPUT# filenum, stringvar

filenum

is the number under which the file was OPENed

stringvar

is the string variable to which the line is to be assigned

Remarks

LINE INPUT# reads all characters in the sequential file up to a CR. It then skips over the CR LF sequence. The next LINE INPUT# reads all characters up to the next CR. If a LF CR sequence is encountered, it is preserved.

LINE INPUT# is especially useful if each line of a data file has been broken into fields, or if a GWBASIC program saved in ASCII format is being read as data by another program. (See "SAVE" in this chapter.)

LINE INPUT# Statement

Example

10 OPEN "O",1,"LIST"
20 LINE INPUT "CUSTOMER ";C\$
30 PRINT #1, C\$
40 CLOSE 1
50 OPEN "I",1,"LIST"
60 LINE INPUT #1, C\$
70 PRINT C\$
80 CLOSE 1
RUN
CUSTOMER? I. JONES 234,4
I. JONES 234,4
Ok

Lists the current program to the screen or a specified file or device.

Syntax

LIST [linenum1] [-[linenum2]] [,{device | filename}]

linenum1

are the line numbers of the first and the last line to be listed. You may use a period (.) for either line number to indicate the current line.

device

is a string expression for the device specification

filename

is a string expression for the file specification

Remarks

If you omit the "device" or "filename", the listing is directed to the screen. You can stop listings directed to either the screen or the printer by pressing CTRL BREAK at any time. You cannot interrupt listings directed to a file or device: the listing continues until the range is exhausted.

If you omit the line range, the entire program is listed.

The syntax allows the following options:

- If "linenum1" is given, followed by a hyphen, that line and all higher numbered lines are listed.
- If only a "linenum1" is given, only that line is listed
- If only "linenum2" is given, all lines from the beginning of the program through the given line are listed.
- If both numbers are specified, the inclusive range is listed.

When you direct a listing to a disk file, the program is saved in ASCII format. You can later use this file with MERGE.

Examples

LIST , LPT1:

List the program to the Line Printer.

LIST 10-90

List lines 10 through 90 to the Screen.

LIST 10-, SCRN:

List lines 10 through last to the Screen.

Lists the current program on the printer. LLIST is usually used in immediate mode.

Syntax LLIST [linenum1] [-[linenum2]]

Remarks The line number ranges are the same as the

LIST command. LLIST assumes a 132-

character-wide printer.

GWBASIC always returns to command level

after an LLIST is executed.

Examples LLIST

Lists a complete program.

LLIST 50

Lists line 50.

LLIST 20-40

Lists lines 20-40

LLIST-150

Lists from the first program line to line 150.



R

Remarks

Loads a program into memory from a file. You can then run the program by nameifying the option R.

Syntax LOAD [filename] [,R]

filename is a string expression which specifies the file to be loaded.

(R) Run is optional. When specified, Run causes the program to begin execution from the first statement after loading.

LOAD deletes all variables and program lines currently residing in memory and closes all open data files before it loads the specified program. If option R is nameified, all open data files are kept open and the program runs after it is loaded.

RUN filename is equivalent to LOAD filename, R.

Examples LOAD "STRTRK",R

Loads and runs the program STRTRK.BAS

LOAD "B:MYPROG"

Loads the program MYPROG.BAS from the disk in drive B, but does not run the program.

7-178

Returns the current position of the file.

Syntax LOC(filenum)

filenum is the number under which the file was OPENed

Remarks

For random disk files, LOC returns the record number just read or written from a GET or PUT Statement. If the file was opened but no disk I/O has yet been performed, LOC returns a '0'.

For sequential files, LOC returns the number of sectors (128-byte blocks) read from or written to the file since it was OPENed. Note that the first sector is read automatically when the file is opened, so LOC will never return less than 1 for a sequential file.

For communications files, LOC returns the number of characters in the input queue waiting to be read. The input queue can hold more than 255 characters (determined by the /C: switch). If there are more than 255 characters in the queue, LOC returns 255. Since a string is limited to 255 characters, you do not need to test for string size before reading data into a string. If fewer than 255 characters remain in the queue, LOC returns the actual count.

Example 100 IF LOC(2)>100 THEN STOP

Moves the graphics cursor to the specified position. LOCATE may also turn the cursor on and off and define the shape and blinkrate of either the overwrite or the user cursor.

Syntax 1 LOCATE [row] [,column] [, [rate] [, [start] [,stop]]]],

Syntax 2 LOCATE [row] [, [column] [, [rate] [, [line] [,map]]]],

is the screen line number. An unsigned integer expression in row the range 1 to 25.

column is the screen column number. An unsigned integer in the range 1 to 40 or 1 to 80, depending on screen width.

is an integer expression in the range 0 to 10 rate

> Turn both the user and the overwrite cursors off 1 Make the specified cursor non-blinking (the 'start' parameter specifies the type of cursor) 2...10 Blink the specified cursor with a period of 'rate' units of 1/18.75 seconds

start

is the cursor starting scanline. It must be an integer expression in the range 0 to 15, or 32 to 47. If 'start' is in the range 0 to 15 the overwrite cursor shape is programmed and a value of rate between 1 to 10 affects the overwrite cursor. If 'start' is in the range 32 to 47, the user cursor shape is programmed, and a non zero value of 'rate' affects the user cursor, not the overwrite cursor. If 'start' is in the range 32 to 47, it is taken to be module 15.

stop

is the cursor stop scanline. It must be a numeric expression in the range $0\ {\rm to}\ 15.$

line

if the value of 'line' is between 50 and 50+M, byte number 'line - 50' of the cursor bitmap for the overwrite cursor is set to 'map'. If the value is between 100 and 100+M, then byte number 'line - 100' of the cursor bitmap for the user cursor is set to 'map'. The value of M is 15 for medium-resolution mode, 7 for high-resolution mode, and 15 for super-resolution mode.

LOCATE (Graphics) Statement

map

if 'line' and 'map' are specified, this value replaces the bitmap for scanline 'line' of the cursor specified by 'rate'. The cursor bitmap is a byte array which is XOR'd with the screen to display the cursor. For medium-resolution mode, each scanline of the cursor is represented by 2 bytes; the low-order byte of each scanline is the left one on the screen. For other modes, there is one byte per scanline. The size of the array is the number of scanlines per row of text times the number of bytes per cursor scanline: this is 8 for high-resolution mode, and 16 for the other modes. Cursor bitmaps are kept separately for screen modes 1, 2, and 3. The cursor state for each mode is restored if another screen mode is selected, and the original mode is reselected. Likewise, separate bitmaps are kept for the insert, overwrite, and user cursors.

Remarks

GWBASIC includes a blinking cursor for graphics mode. The maximum height of this cursor is 8 in modes 1 and 2, and 16 in mode 3. Cursor scanlines are numbered starting with 0 for the top scanline.

The graphics mode as well as in text mode support three different cursors (see the LOCATE text Statement).

The insert-mode cursor will always be a rapidly blinking small triangle at the lower left of the character cell. The overwrite-mode cursor is initially an underline which blinks somewhat more slowly. The user cursor is initially disabled, but its shape array is loaded with OFFH bytes, so that it can easily be made to be any underline or block shape. The shape of the user and overwrite cursors are programmable.

LOCATE,,0 disables both the user and the overwrite cursors. Execution of any graphics statement disables the user cursor (so that the cursor is removed from screen memory while the graphics statement is executed). In this case, the user cursor must be explicitly turned on to be used later on.

Examples 10 LOCATE 5,1,4,2

Moves to line 5, column 1, turns the overwrite cursor on with a blinkrate 4/18.75 seconds, and sets the height of the cursor to 2. (All scanlines of the cursor are initialized to &HFE, so 2 scanlines will appear unless the user has changed the bitmap.)

100 LOCATE ,,,1, &H82 110 LOCATE ,,,2 120 FOR W=1 TO 2000 130 NEXT

Sets the bitmap of the second scanline of the user cursor to binary 10000010, sets its height to 2, and displays the user cursor for a couple of seconds. It will appear as a U-shaped underline like the initial overwrite cursor.

Moves the cursor to the specified position on the active page. LOCATE may also turn the cursor on and off and define the size of either the overwrite or the user cursor.

Syntax

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

row

is the screen line number. A numeric expression returning an unsigned integer in the range 1 to 25.

column

is the screen column number. An unsigned numeric expression returning an unsigned integer in the range 1 to 40 or 1 to 80, depending upon screen width.

cursor

determines whether or not the screen cursors are displayed. A 0 (zero) indicates off, 1 (one) indicates on.

start

is a numeric expression whose integer value represents the cursor starting scanline. If 'start' is in the range 0-31, 'start' and 'stop' will affect the overwrite cursor. If 'start' has a larger value, it will be interpreted modulo 32, and 'start' and 'stop' will change the size of the user cursor.

stop

is a numeric expression whose integer value represents the cursor stop scanline. It must be in the range 0-31.

Remarks

In GWBASIC, there are three cursors: the insert-mode cursor, which appears when insert-mode is in effect, the overwrite cursor, which appears when overwrite mode is in effect (during command entry and input with the INPUT statement), and the user cursor, which appears during program execution when an INPUT statement is NOT being executed. The overwrite cursor is the one which appears most of the time.

The overwrite cursor is initialized to an underline. The insert-mode cursor is a half-height block. The user cursor is initially disabled and undefined. The insert-mode cursor has a fixed size; the sizes of the overwrite and user cursors may be changed.

Following a LOCATE statement, I/O statements to the screen begin placing characters at the specified location. The user cursor is normally off during program execution, but can be turned back on using LOCATE, 1.

Note that "start" and "stop" parameters enable you to define the size of the cursor by indicating the starting and ending scanlines. The scanlines are numbered from 0 at the top of the character position. The bottom scanline is 7 if a color monitor has been installed and if a BW monitor is used. If you specify "start" and omit "stop", this assumes the value of "start". If "start" is greater than "stop", a two-part cursor will be returned.

Normally, GWBASIC will not print to line 25 because of the soft key display. This can be turned off, however, using KEY OFF; then use LOCATE 25,1: PRINT... to display characters on line 25.

Any parameter may be omitted, and will then assume the current value.

Errors

Any values entered outside of the ranges indicated will result in an "Illegal function call" error. Previous values are retained.

Examples

100 LOCATE 1,1

Moves the cursor to the home position in the upper left-hand corner.

200 LOCATE "1

Makes the user cursor visible. Its position remains unchanged.

300 LOCATE ,,0

Turns both the user and overwrite cursors off. This is useful during a program which displays text or graphics and only uses INPUT to input keyboard data (INPUT uses the screen editor).

400 LOCATE 6,1,1,0,7

Moves the overwrite cursor to line 6, column 1. Makes the cursor visible, covering the entire character cell, starting at scanline 0 and ending on scanline 7 (in one of the color modes).

LOCATE ,,1,13

Makes the overwrite cursor visible. Its position remains unchanged. The cursor's shape will be a thin horizontal line at the bottom of the character cell (in monochrome).

LOCATE ,,1,45

Makes the user cursor visible. Its position remains unchanged. The cursor's shape will be a thin horizontal line at the bottom of the character cell (in monochrome). Returns the length of the named file in bytes.

Syntax

LOF(filenum)

Remarks

For disk files LOF returns the actual numbers of bytes allocated to the file.

For communications files LOF returns the amount of file space in the input buffer. The actual value returned is:

buffersize-LOC(filenum)

Where "buffersize" is the size of the communications buffer. It defaults to 256 bytes, but may be changed with the "/C:" option in the GWBASIC command line.

Example

10 OPEN "B:MYFILE" AS #2 20 GET #2,LOF(2)/128

The above statements will get the last record of the file MYFILE (residing on the diskette inserted in drive B) assuming that the file was created with a record length of 128 bytes.

LOG Function

Returns the natural logarithm of a positive argument.

Syntax LOG(numexp)

Remarks LOG is calculated in single precision, unless

you specify the "D" option when you invoke

GWBASIC.

If "numexp" is negative or zero an "Illegal

function call" error is returned.

Example PRINT LOG(45/7)

1.860752

Ok

Returns the current position of the print head within the printer buffer.

Syntax LPOS (printer)

printer is an integer expression whose value (1, 2, or 3) indicates

which printer is to be tested (LPT1:, LPT2:, or LPT3:).

Remarks LPOS does not necessarily give the physical

position of the print head.

Example 150 IF LPOS(1)>60 THEN LPRINT

CHR\$(13)

LPRINT and LPRINT USING

Statement

Prints data on the printer.

Syntax 1 LPRINT[listofexpressions][,|;]

Syntax 2 LPRINT

USINGformatstring; list of expressions[,|;]

list of this list may include numeric and/or string expressions expressions separated by commas or semicolons.

format string is a string expression (usually a string constant or variable)

that is composed of special formatting characters.

Remarks These statements are the same as PRINT and

PRINT USING, except that the output goes to the printer. See PRINT and PRINT USING in

this chapter. LPRINT assumes a 132

character-width printer.

Example **10 A\$ = "For July____"**

20 X = .491

30 LPRINT "Results", A\$,

40 LPRINT X

Ok RUN Results For July___491
The result prints on the line printer.

LSET stores a string value in a random buffer field left justified, or left justifies a string value in a string variable. RSET right justifies the string value.

Syntax

LSETstringvar=stringexp RSETstringvar=stringexp

stringvar

represents either a regular or fielded string variable (i.e., a string variable previously used in a FIELD statement).

stringexp

represents the string to be left or right justified in a given field.

Remarks

If "stringexp" requires fewer bytes than were FIELDed to "stringvar", LSET left-justifies the string in the field, and RSET right-justifies the string. (Spaces are used to pad the extra positions.) If the string is too long for the field, characters are dropped from the right. Numeric values must be converted to strings before they are LSET or RSET. See "MKI\$, MKS\$, MKD\$" in this chapter. See also Chapter 4 for a complete description of random files.

Examples

150 LSET A\$=MKS\$(AMT) 160 LSET D\$=MKI\$(COUNT%)

LSET or RSET may also be used with a nonfielded string variable to left-justify or right-justify a string in a given field. For example, the program lines:

110 A\$ = SPACE\$(20) 120 RSET A\$ = N\$

right-justify the string N\$ in a 20-character field. This is useful when formatting printed output.

Merges the current program with another program previously saved in ASCII format.

Syntax MERGE[filename]

filename specifies the file, and optionally a drive. If the drive is omitted the MS-DOS default drive is assumed.

Remarks This command allows you to merge a program

saved (in ASCII format) on a disk, with the program in memory. MERGE is similar to LOAD, except that the program in memory is not erased before the disk program is loaded. Program lines in the disk program are inserted into the resident program as if they were typed on the keyboard. New lines are added and old lines are updated.

This command allows you to include common subroutines in all of your programs.

Example MERGE "B:ROOT\S1\SUBRTN"

Remarks

As a function, MID\$ returns a substring from a specified string.

Syntax MID\$ (string,start[,length])

string the string from which the substring is taken

start the character position of the beginning of the returned string. It must be an integer expression whose value is >0.

length the length of the returned string. It must be an integer expression from 0 to 255.

The function MID\$ returns a substring taken from a specified string, starting from a specified character position start. The "length" of the substring taken can be specified. If length is omitted or if there are fewer than length characters to the right of the specified character position, all characters to the right of the specified character position are returned. If length is equal to zero, or if start is greater than the length of string, then MID\$ returns a null string.

Also see LEFT\$ and RIGHT\$ functions in this chapter.

Example

Ok 10 A\$ = "HELLO" 20 B\$ = "JOSEPH JOHNNY JIMMY" 30 PRINT A\$;MID\$(B\$,8,6) RUN HELLO JOHNNY Ok As a statement, MID\$ replaces a section of a string with another string.

Syntax

MID\$(string,start[,length])=substring

string

is a string expression whose value is the string from which a section is to be replaced

start

is an integer expression from 1 to 255, whose value specifies the character position where the replacement is to begin; 'start' must be <=LEN (string)

length

is an integer expression from 0 to 255. It represents the length of the section to be replaced with substring.

substring

is a string expression which replaces the characters in 'string', beginning from 'start' position

Remarks

The characters in "string", beginning from "start" position, are replaced by the characters in "substring". The optional "length" refers to the number of characters from "substring" that will be used in the replacement. If "length" is omitted, all of the characters of "substring" are used.

The replacement of characters never goes beyond the original length of "string."

Example

Ok 10 A\$ = "AVIGNON, FRANCE" 20 MID\$(A\$,10) = "ROUBAIX" 30 PRINT A\$ RUN AVIGNON, ROUBAIX Ok

Note that the original string length was not changed.

Makes a new directory on a specified disk.

Syntax MKDIR pathname

pathname is a string expression specifying the name of the directory to

be created

Example Assume that our current directory is the root:



To create a sub-directory MARKETING from the root on the current drive, enter:

MKDIR "MARKETING"

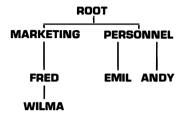
To create a sub-directory called FRED under the directory MARKETING, enter:

MKDIR "MARKETING\FRED"

To create a sub-directory called WILMA under the directory FRED, enter:

MKDIR "MARKETING\FRED\WILMA"

The resulting structure will be:



Make a string value from a numeric value.

Syntax 1 MKI\$(integer expression)

Syntax 2 MKS\$(single precision expression)

Syntax 3 MKD\$(double precision expression)

Remarks Any numeric value that is placed in a random file buffer with an LSET or RSET statement must be converted to a string.

MKI\$ converts an integer to a 2-byte string. MKS\$ converts a single-precision number to a 4-byte string. MKD\$ converts a double-precision number to an 8-byte string.

See also "CVI, CVS, CVD Functions" in this chapter.

Example 90 AMT = (K + T)

100 FIELD #1,8 AS D\$,20 AS N\$

110 LSET D\$ = MKD\$ (AMT)

120 LSET N\$ = A\$

130 PUT #1

Changes the name of a disk file.

Syntax NAME filename AS filename

filename is a string expression which specifies the file to be renamed.

The file must exist on the specified drive. If the drive is not specified the MS-DOS default drive is assumed. The file extension does not default to BAS. Pathnames are not allowed.

filename the new name. The new name should not already exist for

another file.

Remarks After a NAME command, the file exists on the

same disk, with the new name. The area allocated to the file is not changed. A file may not be renamed with a new drive designation. If this is attempted, a "Rename across disks"

error is generated.

NAME Command

Example

Ok NAME "B:GRAPH.BAS" AS "GRAPH1.BAS" Ok

In this example, the file that was formerly named GRAPH.BAS on the diskette in drive B: will now be named GRAPH1.BAS.

Deletes the current program and clears all variables, so that you can enter a new program.

Syntax

NEW

Remarks

NEW is entered at command level to clear memory before entering a new program. GWBASIC always returns to command level after a NEW command is executed. NEW closes all data files and switches off the trace flag in the same way as TROFF.

Returns a string which is the octal value of the decimal argument.

Syntax OCT\$(numexp)

numexp is a numeric expression from 32768 to 65535, which is

rounded to the nearest integer before OCT\$ is evaluated.

Remarks When "numexp" is negative, the two's

complement form is used.

Example PRINT OCT\$(24) 30

Ok

See the HEX\$ function in this chapter for

details on hexadecimal conversion.

Specifies the first line number of a trap subroutine to be activated as soon as characters arrive in the communications buffer.

The ON COM(n) GOSUB statement is only used in a program.

Syntax ON COM(n) GOSUBlinenum

n is the number of the communications channel (1, 2, 3, or 4)

linenum is the first line number of the trap routine. A line number of 0

disables the communications event trap.

To Enable or Disable COM Trapping

COM(n) ON COM(n) trapping is enabled

COM(n) OFF COM(n) trapping is disabled

an error trap occurs all event trapping will be disabled (including

ERROR, COM(n), KEY(n), PLAY(n))

COM(n) STOP COM(n) trapping will be suspended, i.e., the GOSUB is

not performed, but is performed as soon as a COM(n) ON

statement is executed.

Remarks

To avoid recursive traps a COM(n) STOP is automatically executed when the trap occurs.

A RETURN from the trap routine automatically performs a COM(n) ON (unless a COM(n) OFF was performed within the trap routine). The RETURN line form may also be used. Use this form with care because any other active GOSUB, WHILEs, or FORs remain active and errors (such as "FOR without NEXT") may result.

Typically, the COM trap routine reads an entire message from the COM port before returning. Do not use the COM trap for single character messages since, at high baud rates, the overhead of trapping and reading for each individual character may cause the COM interrupt buffer to overflow.

Example

This example sets up a trap routine for the second communications channel at line 1000.

When a communications event is trapped, program flow branches to the subroutine defined by the ON COM(n) GOSUB statement.

100 ON COM(2) GOSUB 1000 110 COM(2) ON : : 1000 REM COM activity

: 1050 RETURN 200

ON ERROR GOTO

Statement

Enables error trapping and specifies the first line number of a subroutine to be executed if an error occurs.

The ON ERROR GOTO statement is only used in a program.

Syntax

ON ERROR GOTO linenum

linenum

is the first line number of the error trapping routine

To Enable or Disable ERROR Trapping

ON ERROR GOTO n

ERROR trapping is enabled

ON ERROR GOTO 0

ERROR trapping is disabled. Subsequent errors print an error message and halt execution.

Remarks

If ERROR trapping is enabled and a GWBASIC error (or a user defined error) is found, the ON ERROR GOTO line will be executed and the corresponding routine activated. The ERL and ERR functions are usually used in IF...GOTO...ELSE or IF...THEN...ELSE statements to direct program flow within an error trapping routine.

It is recommended that the error trapping routine execute an ON ERROR GOTO 0 if an error is found for which there is no recovery action. (In this case the standard error message will be displayed and execution will stop.) The RESUME statement resumes execution after the error handling routine has been entered (see the RESUME statement in this chapter). If a GWBASIC error (or a user-defined error) is found, during the execution of an error trapping routine the associated error message is displayed and execution terminates.

Note: Error trapping does not occur within the error trapping routine.

Example

10 ON ERROR GOTO 1000 20 INPUT R 30 IF R=0 THEN ERROR 300

-

100 IF ERR = 300 THEN PRINT "RADIUS NEGATIVE OR ZERO" 110 IF ERL = 30 THEN RESUME 20 120 ON ERROR GOTO 0

ON...GOSUB and ON...GOTO

Statements

ON...GOSUB calls one of several specified subroutines, depending on the value of a specified expression. ON...GOTO branches like GOSUB but does not return from the branch.

Syntax 1 ON numexp GOSUB linenum [,linenum]...

Syntax 2 ON numexp GOTO linenum [, linenum]...

numexp is a numeric expression (from 0 to 255) which determines which line number in the list to use for branching. If 'numexp' is not an integer, it is rounded up to an integer.

linenum is the line number to which the branch is made

Remarks

In the ON...GOSUB statement, each line number in the list must be the first line number of a subroutine. If the value of "numexp" is 1 the subroutine at the first line number in the list will be called; a value of 2 causes the subroutine at the second line number in the list to be called; and so on. If the value of "numexp" is zero or greater than the number of items in the list (but less than or equal to 255), GWBASIC continues with the next executable statement. If the value of "numexp" is negative or greater than 255, an "Illegal function call" error occurs.

Example 100 ON L-1 GOTO 150,300,320,390

Statement

Specifies the first line number of a subroutine to be executed when a specified key is pressed.

The ON KEY(n) GOSUB statement is only used in a program.

Syntax

ON KEY (n) GOSUB linenum

n

is an integer from 1-20. It specifies the key to be be trapped as follows:

1-10	function keys F1 to F10
11	Cursor Up
12	Cursor Left
13	Cursor Right
14	Cursor Down
15-20	Keys defined in the form:
	KEY n, CHR\$(shift)+CHR\$(scan code)
(C () TED37	0

(See "KEY Statement" in this chapter)

linenum

is the list line number of the routine that is to be performed when the specified function or cursor direction key is pressed. A line number of 0 disables the event trap.

To Enable or Disable KEY(n) Trapping

KEY(n) ON KEY(n) trapping will be

enabled

KEY(n) OFF KEY(n) trapping will be

disabled

an error trap occurs

all event trapping will be disabled including ERROR, COM(n), KEY(n), PLAY(n)

KEY(n) STOP KEY(n) trapping will be

suspended, i.e., the GOSUB is not performed, but it will be performed as soon as a KEY(n) ON is executed.

Remarks

If KEY(n) trapping is enabled and key n was pressed ON KEY(n) GOSUB is executed and the corresponding routine activated.

To avoid recursive traps a KEY(n) STOP is automatically executed, when the trap occurs. A RETURN from the trap routine automatically performs a KEY(n) ON (unless a KEY(n) OFF was performed within the trap routine).

The RETURN line form may also be used. Use this form with care, because any other active GOSUBs, WHILEs, or FORs remain active, and errors may result.

You cannot use the INPUT or INKEY\$ statements to find out which key caused the trap. If you wish to assign different functions to particular keys, you must set up a different subroutine for each key, rather than assigning the various functions within a single subroutine.

Example

10 KEY 4,"SCREEN 0,0" 'assigns softkey 4 20 KEY(4) ON 'enables event trapping 70 ON KEY(4) GOSUB 200

key 4 pressed

200 'Subroutine for screen 250 RETURN

In the above, the programmer has overridden the normal function associated with function key 4, and replaced it with "SCREEN 0,0", which will be displayed whenever that key is pressed.

Example

100 KEY 15, CHR\$(&HO4) + CHR\$(83)
105 REM ** Key 15 now is CTRL DEL **
110 KEY(15) ON
1000 PRINT "If you want to stop processing for a break"
1010 PRINT "press the CTRL key and the DEL key at the"
1020 PRINT "same time."
1030 ON KEY(15) GOSUB 3000.

Operator presses CTRL DEL

3000 REM ** Suspend processing loop.
3010 CLOSE #1
3020 RESET
3030 CLS
3035 PRINT "Enter CONT to continue."
3040 STOP
3050 OPEN "A", #1, "ACCOUNTS.DAT"
3060 RETURN

In the above, the programmer has enabled the CTRL DEL key to enter a subroutine which closes the files and stops program execution until the operator is ready to continue.

Specifies the first line number of a subroutine to be executed when the music buffer contains fewer than "n" notes. This permits continuous background music during program execution.

The ON PLAY(n) GOSUB statement is only used in a program.

Syntax

ON PLAY(n) GOSUB linenum

n

is an integer expression in the range 1 to 32. Values outside this range result in an "Illegal function call" error.

linenum

is the first line number of the associated trap routine. A line number of 0 disables play trapping.

To Enable or Disable PLAY(n) Trapping

PLAY ON PLAY(n) trapping is enabled

PLAY OFF PLAY(n) trapping is disabled

an error trap occurs

all event trapping is disabled

PLAY STOP PLAY(n) trapping is

> suspended, i.e., the GOSUB is not performed, but it is performed as soon as a PLAY ON is executed.

Remarks

If PLAY(n) trapping is enabled, and the background music buffer has gone from 'n' to 'n-1' notes, then the ON PLAY(n) GOSUB line is executed, and the corresponding routine activated. To avoid recursive traps, a PLAY STOP is automatically executed when the trap occurs

A RETURN from the trapping subroutine automatically performs a PLAY(n) ON (unless an explicit PLAY(n) OFF was performed within the trap routine). The "RETURN linenum" form may also be used. Use this form with care, because any other active GOSUBs, WHILEs, or FORs will remain active, and errors (such as "FOR without NEXT") may result.

If PLAY(n) trapping is enabled, and the background music buffer is empty, no PLAY(n) trapping routine is executed.

Note:

- A PLAY event trap is only effective when playing Background Music (PLAY "MB...").
 PLAY event traps have no effect when running in Music Foreground (PLAY "MF...").
- A PLAY event trap is ineffective if the Music Background buffer is already empty when a PLAY ON is executed.
- Care should be taken in selecting values for "n." If "n" is a large number, event traps occur frequently enough to reduce program execution speed.

Example

10 PLAY ON 20 ON PLAY(8) GOSUB 1000 1000 'SUB PLAY(8) TRAP 1050 RETURN Sets up a line number for BASIC to trap to when one of the joystick buttons (triggers) is pressed.

Syntax

ON STRIG (n) GOSUB line

n

may be 0, 2, 4, or 6, and indicates the button to be trapped as follows:

- 0 button A1
- 2 button B1
- 4 button A2
- 6 button B2

line

is the line number of the trapping routine. If line is 0, trapping of the joystick button is disabled.

Remarks

The ON STRIG(n) statement causes a program to branch to a specified routine when a specific joystick button is pressed. A STRIG(n) ON statement must first be executed in order for an ON STRIG (n) statement to be active.

The trap routine passes control back in one of two ways. RETURN branches to the program line at which the interrupt was detected. RETURN n branches to line n.

To avoid recursive traps a STRIG(n) STOP is automatically executed when the trap occurs.

A RETURN from the trap routine automatically performs a STRIG(n) ON (unless a STRIG(n) OFF was performed within the trap routine). The RETURN line form may also be used.

Example

This is an example of a trapping routine for the button on the first joystick.

100 ON STRIG(0) GOSUB 2000 110 STRIG(0) ON

•

2000 REM subroutine for 1st button

•

2100 RETURN

Causes an event trap every 'n' seconds.

The ON TIMER (n) GOSUB statement is only used in a program.

Syntax

ON TIMER (n) GOSUB linenum

n

is an integer expression in the range 1 through 86400 (1 second through 24 hours). Values outside this range will result in an "Illegal function call" error.

linenum

is the first line number of the TIMER trapping routine.

Remarks

If event trapping is enabled, and if "line" in the ON TIMER GOSUB statement is not zero, GWBASIC checks between statements to see if the time has been reached. If it has, a GOSUB is performed to the specified line.

If a TIMER OFF statement has been executed the GOSUB is not performed and is not remembered.

If a TIMER STOP statement has been executed the GOSUB is not performed, but is performed as soon as a TIMER ON statement is executed.

When an event trap occurs (i.e., the GOSUB is performed), an automatic TIMER STOP is executed so that recursive traps cannot take place. The RETURN from the trapping subroutine automatically performs a TIMER ON statement unless an explicit TIMER OFF was performed inside the subroutine.

The RETURN "linenum" form of the RETURN statement may be used to return to a specific line number from the trapping subroutine. Use this type of return with care, however, because any other GOSUBs, WHILEs, or FORs that were active at the time of the trap remain active and errors such as "FOR without NEXT" may result.

Example

To display the time of day on line 1 every minute:

10 ON TIMER (60) GOSUB 1000 20 TIMER ON

•

1000 OLDROW = CSRLIN 'save current row

1010 OLDCOL = POS(0) 'save current column

1020 LOCATE 1,1 : PRINT TIME\$

1030 LOCATE OLDROW, OLDCOL 'restore row and column

1040 RETURN

Allows I/O to a file or device. OPEN is usually used in a program.

Syntax 1 OPEN {device|filename} [FOR model] AS [#lfilenum [LEN = recl]

Syntax 2 OPEN mode2, [#]filenum, {device|filename} [,recl]

device is a string expression which specifies the device to be opened

filename is a string expression which specifies the file to be opened. It may optionally include a device.

> is a literal string not enclosed in quotation marks. It determines the initial file pointer position and the action to be taken if the file does not exist. The valid modes and actions taken are:

INPUT Specifies sequential input mode. Positions the pointer to the beginning of an existing file. A "File not found" error is given if the file

does not exist.

OUTPUT Specifies sequential output mode. Positions

the pointer to the beginning of the file. If the file does not exist, one is created.

APPEND Specifies sequential output after the last record on the file. Positions the pointer to the end of

the file. If the file does not exist, one is created.

If the FOR 'model' clause is omitted, the initial position is at the beginning of the file. If the file is not found, one is created. This is the Random I/O mode. That is, records may be read or written at any position within the

file.

mode1

7-221

OPEN Statement

filenum

is an integer expression returning a number in the range 1 through 15. The number is used to associate an I/O buffer with a disk file or device. This association exists until a CLOSE or CLOSE 'filenum' statement is executed. The file is referred by this number in any I/O statement.

record length

is an integer expression from 2 to 32767. This value sets the record length to be used for random files (see the FIELD statement). If omitted, the 'record length' defaults to 128 byte records. The specified 'record length' may not be greater than the value specified by the '/S:' switch on the GWBASIC command. GWBASIC will ignore this option if it is used to OPEN a sequential file.

mode2

is a string expression whose first character is one of the following:

- O Specifies sequential output mode
- I Specifies sequential input mode
- R Specifies random input/output mode

Remarks

A disk file must be opened before any disk I/O operation can be performed on that file. OPEN allocates a buffer for I/O to the file or device and determines the mode of access that will be used with the buffer. The "filenum" parameter specifies the number which will be associated with the file as long as it is open and is used by other I/O statements to refer to the file or device.

For each device or file, the following OPEN modes are allowed:

KYBD: INPUT only
SCRN: OUTPUT only
LPT1: OUTPUT only
LPT2: OUTPUT only
LPT3: OUTPUT only

COM1:, COM2:, INPUT, OUTPUT, or random only

COM3:, and COM4: Disk files allow all modes The GWBASIC file I/O system allows you to utilize user installed devices.

Character devices are opened and used in the same manner as disk files except that characters are not buffered by GWBASIC as they are for disk files. The record length is set to one.

GWBASIC only sends a carriage return as end of line. If the device requires a LF (line feed), the driver must provide it.* When writing device drivers, keep in mind that GWBASIC users want to read and write control information. Writing and reading of device control data is handled by the GWBASIC IOCTL statement and IOCTL\$ function.

Rules

- If you enter a value outside of the corresponding range, an "Illegal function call" error is returned, and the file will not be opened.
- If the file is opened for INPUT, attempts to write to the file result in a "Bad File Mode" error. If a file that is opened for input does not exist, a "File not found" error message is displayed.

^{*}The exception to this is output sent to a printer (LPT1, LPT2, or LPT3), where each line ends with a linefeed unless the printer is opened as a random file and WIDTH is set to 255.

- When a file is opened for APPEND, the pointer position is initially at the end of the file and the record is set to the last record of the file. PRINT# or WRITE# then extends the file
- If the file is opened for OUTPUT or APPEND, attempts to read the file result in a "Bad File Mode" error.
- If you open a file which does not exist for output, append, or random access, that file is automatically created.
- A file can be opened for sequential input or random access on more than one file number at a time. A file may NOT be opened for OUTPUT or APPEND on more than one file number at a time.

Since you can reference the same file in a subdirectory via different pathnames, it is impossible for GWBASIC to know that it is the same file simply by looking at the pathname. For this reason, GWBASIC does not let you open the file for OUTPUT or APPEND if it is on the same disk, even if the pathname is different.

Examples

10 OPEN "!",2,"!NVEN"

10 OPEN "MAILING.DAT" FOR APPEND AS 1

If you write and install a device called FOO, then the OPEN statement can be:

10 OPEN "DEVFOO" FOR OUTPUT AS #1

To open the printer for output, you could use the line:

100 OPEN "LPT:" FOR OUTPUT AS #1

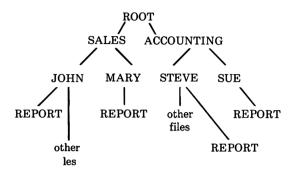
which uses the GWBASIC device driver. You can use part of a pathname:

100 OPEN "DEVLPT1" FOR OUTPUT AS #1

This statement uses the MS-DOS device driver.

Examples

Using the following tree structure:



If MARY is your current directory, then:

OPEN "REPORT"...
OPEN "\SALES\MARY\REPORT"...
OPEN "..\MARY\REPORT"...
OPEN "..\.\MARY\REPORT"...

all refer to the same file.

Possible Errors

- "Bad File name"
- "Bad File number"
- "Bad File Mode"
- "Too many files" Too many files are open. (See the '/F:' switch in the GWBASIC command line.)
- "File not found" If a file opened for input does not exist, a "File not found" error occurs.
- "Device not available" You have attempted to open either a directory, or a non-existent device.
- "File already open"
- "Device I/O error" Reception error. Usually caused by an incorrectly written device driver (user-installed).
- "Illegal function call" Usually caused by an excessive record length. (See the '/S:' switch in the GWBASIC command line.)

Opens a communications file.

Syntax OPEN "COMn:[speed][,parity][,data][,stop]

[,RS][,CS[n]][,DS[n]][,CD[n]][,BIN][,ACS]

[,LF]" AS [#] filenum [LEN = m]

n is 1, 2, 3, or 4. It specifies the number of the Asynchronous Com-

munications Adapter.

speed is an integer constant that sets the transmit/receive baud rate

to one of the following speeds: 75, 110, 150, 300, 600, 1200, 1800,

2400, 4800, or 9600. The default is 300 bps.

parity designates the parity of the device to be OPENed:

E EVEN (default) M MARK (1) O ODD S SPACE (0)

N NONE

data designates the number of data bits. Valid entries are 5, 6, 7, or

8. The default is 7.

stop designates the number of stop bits. Valid entries are 1, 1.5, or

2. Default is 2 for 75 bps and 110 bps, 1 for all other speeds.

RS suppresses the request-to-send (RTS) control character. If RS is

not used, RTS will be sent automatically as soon as the

communications line is OPENed.

CS[n]

controls clear-to-send (CTS). [n] specifies the number of milliseconds before the host times out. [n] may range from 0 to 65535. The default is 1000. If you do not specify [n] or [n] =0, the line status is not checked.

Subsequent communications statements will fail if you do not include CS[n].

DS[n]

controls data-set-ready (DSR). [n] specifies the number of milliseconds before the host times out. [n] may range from 0 to 65535. The default is 1000. If you do not specify [n], the line status is not checked.

Subsequent communications statements will fail if you do not include DS[n].

CD[n]

controls carrier-detect (CD). [n] specifies the number of milliseconds before the host times out. [n] may range from 0 to 65535. The default is 1000. If you do not specify [n], the line status will not be checked.

CD is also referred to as the "received line signal detect" message.

BIN

opens the device in binary mode. BIN is selected by default, unless ASC is specified. See "Remarks" for further discussion of BIN.

ASC

opens the file in ASCII mode. See "Remarks" for further discussion of ASC.

LF

specifies that a linefeed is to be sent after a carriage return (see "Remarks").

filenum

is an integer expression returning a valid file number which is associated with the file while it is OPEN.

M

is the maximum number of bytes that can be read from or written to the communications buffer with GET or PUT. The default is 128.

Remarks

The OPEN COM statement must be executed before a device can be used for RS232 communications.

A COM device may be OPENed to only one file number at a time.

Any syntax errors in the OPEN COM statement will result in a "Bad File name" error. An indication as to which parameter is in error is not given.

A "Device Timeout" error occurs if Data Set Ready (DSR) is not detected.

The "speed", "parity", "data", and "stop" options must be listed in the order shown in the above syntax. The remaining options may be listed in any order, but you must list them after the "speed", "parity", "data", and "stop" options.

LF allows communication files to be printed on a serial line printer. When LF is specified, a linefeed character (0AH) is automatically sent after each carriage return character (0DH). This includes the carriage return sent as a result of the width setting. INPUT# and LINE INPUT#, when used to read from a COM file that was opened with the LF option, stop when they see a carriage return. The linefeed is always ignored.

The LF option is superseded by the BIN option.

In the BIN mode, tabs are not expanded to spaces, a carriage return is not forced at the end-of-line, and CTRL Z is not treated as end-of-file. When the channel is closed, CTRL Z is sent over the RS232 line. The BIN option supersedes the LF option.

In ASC mode, tabs are expanded, carriage returns are forced at the end-of-line, CTRL Z is treated as end-of-file, and XON/XOFF protocol (if supported) is enabled. When the channel is closed, CTRL Z will be sent over the RS232 line.

Example 10 OPEN "COM1:9600,N,8,1,BIN" AS #2

will open communications channel 1 at a speed of 9600 baud with no parity bit, 8 data bits, and stop bit. Input/Output will be in the binary mode. Other lines in the program may now access channel 1 as file number 2.

Defines the minimum value for array subscripts.

Syntax

OPTION BASE n

n

is an integer expression and may be 1 or 0

Remarks

The default base is 0. If the statement:

OPTION BASE 1

is executed, the lowest value an array subscript may have is 1.

A CHAINed program may have an OPTION BASE statement if no arrays are passed. Otherwise, the CHAINed program will inherit the OPTION BASE value of the chaining program.

Possible Errors

The OPTION BASE statement must be coded before definition or usage of arrays. A "Duplicate Definition" error occurs when the base value is changed when arrays are in existence.

Transmits a byte to an output port.

Syntax OUT port, byte

port is an integer expression in the range 0 through 65535 and

represents a port number

byte is an integer expression in the range 0 through 255 and

represents the data to be transmitted

Remarks OUT is the complementary statement to the

INP statement.

If "port" or "byte" is outside the specified

range, an "Illegal function call" error is

returned.

Example 100 OUT 1234,255

Paints an enclosed area on the screen with a specified color (Graphics Mode only).

Syntax

PAINT [STEP] (x,y)[,[paint][,[border] [,background]]

x.v

are the coordinates, either absolute or relative, of a point where painting is to begin. Painting should always start on a non-border point.

paint

is a numeric or string expression. If it is a numeric expression in the range 0 to 3, it represents the color number to be used for painting (see the COLOR graphics statement for the current screen mode, for details). If it is a string expression then 'tiling' is performed. Tiling is described in detail later in this section.

border

is an integer expression in the range 0 to 3. It identifies the border color of the figure to be filled.

background

is a string expression returning one character, used in "paint tiling."

Remarks

The PAINT statement will fill in an arbitrary figure, with edges of border color with the specified paint color. The paint color will default to the graphics foreground color if not given, and the border color defaults to the paint color.

For example, in medium resolution you can fill in a circle of color 1 with color 2. Visually, this could mean a red ball with a green border (if palette 0 were selected).

Since there are only two colors in highresolution and super-resolution mode, this means "whiting out" an area until white is encountered, or "blacking out" an area until black is encountered.

PAINT must start on a non-border point; otherwise PAINT will have no effect.

If the specified point already has the color "border", the PAINT will have no effect.

PAINT can fill any figure, but PAINTing "uneven" edges on very complex figures may result in an "Out of Memory" error. If this happens, you must use the CLEAR statement to increase the amount of stack space available.

Tiling

A figure may be "tiled" using the paint parameter as a string expression of the form:

CHR\$(&Hnn) + CHR\$(&nn) + CHR\$(&Hnn)...

where the two hexadecimal numbers (&Hnn) correspond to 8 bits. The tile mask is always 8 bits wide and the string expression may be from 1 to 64 bytes long.

The structure of the string expression appears as follows:

The tile pattern is replicated uniformly over the entire screen.

Each byte in the tile string masks 8 bits along the x axis when plotting points. Each byte of the tile string is rotated as required to align the y axis such that:

tile_byte_mask = y MOD tile_length

Since there is only one bit per pixel in highand super-resolution modes (SCREEN 2 and 3), a point is plotted at every position in the bit mask which has a value of 1. In high- and super-resolution mode, the screen can be painted with 'x's by the following statement:

Syntax

```
PAINT (320,100),CHR$(&H81) +
CHR$(&H42) + CHR$(&H24) + CHR$(&H18) +
CHR$(&H18) + CHR$(&H24) + CHR$(&H42) +
CHR$(&H81)
```

This pattern appears on the screen as:

x increases ->

```
0.0
      |1|0|0|0|0|0|0|1|
                     CHR$ (&H81) Tile byte 0
0,1
      0100001010
                     CHR$ (&H42) Tile byte 1
      0001001100
                     CHR$ (&H24) Tile byte 2
0,2
0,3
      000111000
                     CHR$ (&H18) Tile byte 3
      0001110000
0,4
                     CHR$ (&H18) Tile byte 4
0,5
      |0|0|1|0|0|1|0|0|
                     CHR$ (&H24) Tile byte 5
0,6
      |0|1|0|0|0|0|1|0|
                     CHR$ (&H42) Tile byte 6
0.7
      |1|0|0|0|0|0|0|1| CHR$ (&H81) Tile byte 7
```

Since there are 2 bits per pixel in mediumresolution mode (SCREEN 1), each byte of the tile pattern only describes 4 pixels. In this case, every 2 bits of the tile byte describes 1 of the 4 possible colors associated with each of the 4 pixels to be put down.

If "background" color is omitted, the default value is CHR\$(0). When supplied, "background" specifies the "background tile" pattern or color byte to skip when checking for border termination.

It may occasionally be necessary to tile paint over an area that is the same color as two consecutive lines in the tile pattern. Normally, paint quits when it encounters two consecutive lines of the same color as the point being set (the point is surrounded). It would not be possible to draw alternating blue and red lines on a red background without this parameter.

Paint would stop as soon as the first red pixel was drawn. Specifying red [CHR\$(&HAA)] as the background color, allows the red line to be drawn over the red background.

You cannot specify more than two consecutive bytes in the tile string that match the background color. Specifying more than two will result in an "Illegal function call" error.

PAINT Statement

Example

10 SCREEN 1 20 COLOR 0,0,1,0 30 CLS 40 CIRCLE (256,128),130,2 50 PAINT (256,128),1,2 60 LINE (251,123)-STEP(10,10),0,BF

Statement 10 selects Medium Resolution Mode. Statement 20 selects black for color number 0, palette 0 (green, red, yellow), green as graphics foreground, black as graphics background. Statement 30 clears the screen with the background color (in this case black). Statement 40 draws a red circumference with a radius of 130 whose center is (256,128). Statement 50 paints the circle green. Statement 60 draws a black filled-in box in the middle of the circle.

Returns the byte read from the specified memory location.

Syntax PEEK (offset)

offset is a numeric expression returning an integer in the range 0 to 65535. It indicates the address of the memory location from which a byte will be returned. It is the offset from the current segment, which was defined by the last DEF SEG

statement.

Remarks The returned value is an integer in the range 0

to 255.

If "offset" is outside the specified range, an "Illegal function call" error is returned.

PEEK is the complementary function of the

POKE statement.

Example A = PEEK(&H5A00)

Plays music in accordance with a string which specifies the notes to be played, and the way in which the notes are to be played.

Syntax PLAY stringexp

stringexp is a string expression containing a series of single-character commands

Remarks

PLAY uses a concept similar to that in DRAW (see the DRAW statement in this chapter) by embedding a Music Macro Language into one statement. A set of subcommands, used as part of the PLAY statement, specifies the

particular action to be taken.

The single-character commands available for the PLAY string are as follows:

Plays the specified note in the current octave. The optional suffixes (#) or (+) produce a sharp note; suffix (-) produces a flat note. Sharp and flat notes that do not correspond to a black key on a piano are not allowed.

Sets the octave number, from 0 to 6.

Increments the octave and plays note n. The octave is progressively incremented, each time note n is played, until octave 6 is reached. Note n is subsequently played at octave 6.

On

A-G

<n Decrements the octave and plays note n. The octave is progressively decremented, each time note n is played, until octave 0 is reached. Note n is subsequently played at octave 0.</p>

Nn

Pn

Plays one of 84 notes within the 7 possible octaves. The Nn parameter ranges from 0 to 84; 0 indicates a rest. This command is an alternative to specifying notes using the note name (A-G) and octave number commands.

Specifies a pause. The n parameter ranges from 1 to 64 and corresponds to the length of each note, set by Ln.

Ln Sets the length of each note. The n parameter ranges from 1 to 64, where n=1 is equivalent to a whole note; n=4 is equivalent to a quarter note, etc. The length may also follow the note when a change of length only is required for a particular note. In this case, A16 is equivalent to L16A.

A dot or period after a note causes it to be played at 1½ times the specified length. Multiple periods may appear after a note, and the length is adjusted accordingly; e.g., A. is 3/2, A. is 9/4, etc. Periods may appear after a pause, and the pause length is adjusted accordingly.

Tn	Sets the "tempo", or number of quarter notes, in one minute. The n parameter ranges from 32 to 255, with a default value of 120.
MF	Sets Music Foreground. Music (PLAY statement) and SOUND are to run in Foreground. Each successive note does not start until the preceding note has finished. Music foreground is the default setting.
МВ	Sets Music Background. Music (PLAY statement) and SOUND are to run in Background. The GWBASIC program continues as music plays in the "background." Up to 32 notes (or rests) can be played in the background at a time.
MN	Sets "music normal", so that each note will play 7/8 of the time determined by length (L).
ML	Sets "music legato", so that each note will play the full period set by length (L).
MS	Sets "music staccato", so that each note will play 3/4 of the time set by length (L).
X variable	Executes the specified variable string.

Remarks

The "n" parameter may be constant or variable, where a variable is written as: "+variable;." The semicolon is necessary when a variable is used in this way, or when the X command is used, but it is not allowed after MF, MB, MN, ML, or MS. In all other cases, a semicolon is optional between commands.

When the X command is used, VARPTR\$ (variable), may be substituted for "variable;" as in the example below.

Example

100 PLAY "02 L4 C P1 C P2 C P4"

200 PLAY "XB\$;XC\$;XD\$;"

300 PLAY "XM\$;" or

300 PLAY "X" + VARPTR\$(M\$)

Returns the number of notes remaining in the

music background buffer.

Syntax PLAY (dummy)

dummy is a dummy argument. Any value may be supplied.

Remarks If the program is running in Music

Foreground mode, PLAY(n) returns 0.

If the program is running in Music Background mode, PLAY(n) returns the number of notes currently in the Music Background buffer. The maximum value that

PLAY(n) may return is 32.

Example | **IF PLAY(0)** = **6 GOTO 500**

PLAY ON enables PLAY(n) trapping. PLAY OFF disables PLAY(n) trapping. PLAY STOP suspends PLAY(n) trapping.

Syntax

PLAY {ON | OFF | STOP}

Remarks

PLAY ON, PLAY OFF, PLAY STOP are used in conjunction with the ON PLAY(n) statement.

If a PLAY OFF statement has been executed the GOSUB is not performed and is not remembered.

If a PLAY STOP statement has been executed the GOSUB is not performed, but will be performed as soon as a PLAY ON statement is executed.

When an event trap occurs (i.e., the GOSUB is performed), an automatic PLAY STOP is executed so that recursive traps cannot take place. The RETURN from the trapping subroutine will automatically perform a PLAY ON statement unless an explicit PLAY OFF was performed inside the subroutine.

The RETURN "linenum" form of the RETURN statement may be used to return to a specific line number from the trapping subroutine. Use this type of return with care, however, because any other GOSUBs, WHILEs, or FORs that were active at the time of the trap will remain active, and errors such as "FOR without NEXT" may result.

Converts physical coordinates to world coordinates or vice versa (Graphics Mode only.)

Syntax

PMAP (coordinate,n)

coordinate

is a numeric expression specifying the x or y coordinate of the point to be mapped.

n

is an integer in the range 0 to 3: 0 maps the world coordinate x to the physical coordinate x 1 maps the world coordinate y to the physical coordinate y 2 maps the physical coordinate x to the world coordinate x 3 maps the physical coordinate y to the world coordinate y

Remarks

The four PMAP functions allow you to find equivalent point locations between the world coordinates created with the WINDOW statement and the physical coordinate system of the screen or viewport as defined by the VIEW statement.

Examples

Given a defined **WINDOW SCREEN** (80,100)-(200,200) the upper left coordinate of the window is (80,100) and the lower right is (200,200). The screen coordinates are (0,0) in the upper left hand corner and (639,199) in the lower right. Then:

X = PMAP(80,0)

returns the screen x coordinate of the window x coordinate 80: 0

Y = PMAP(200,1)

returns the screen y coordinate of the window y coordinate 200: 199

X = PMAP(619,2)

returns the "world" x coordinate that corresponds to the screen or viewport x coordinate 619: 199

The PMAP function in the statement:

Y = PMAP(100,3)

returns the "world" y coordinate that corresponds to the screen or viewport y coordinate 100: 140

With two arguments (x,y) returns the color number of a pixel on the screen. If one argument (n) is given, returns current graphics coordinate. (Graphics Mode only).

Syntax

POINT (n)

x,y

are the absolute coordinates of the pixel to be referenced. If the point is out of range, the value -1 is returned.

n

- 'n' may have the values 0, 1, 2, or 3:
- 0: Returns the current physical x coordinate
- 1: Returns the current physical y coordinate
- 2: Returns the current world x coordinate if a WINDOW statement has been used, other-wise returns the same value as the POINT(0) function.
- 3: Returns the current world y coordinate if WINDOW is active, otherwise returns the same value as the POINT(1) function.

Remarks

v1 + POINT(x,y)

returns the color number of the specified pixel into the integer variable v1.

v2+POINT(n)

returns the specified coordinate of the current point into the single (or double) precision variable v2.

Examples

10 SCREEN 0,0 20 FOR K = 0 TO 3 30 PSET (10,10),K 40 IF POINT(10,10) < > K THEN PRINT "Broken Basic!" 50 NEXT

10 SCREEN 2
20 IF POINT(I,I) <> 0
THEN PRESET (I,I) ELSE PSET (I,I)
30 'Invert current state of point(I,I)
40 PSET (I,I),1-POINT(I,I)
50 'Another way to invert a point, if the
55 'system is B/W

10 SCREEN 1
20 LET C = 3
30 PSET (10,10),C
40 IF POINT(10,10) = C
THEN PRINT "This point is color ";C

Writes a byte into a memory location.

Syntax POKE offset, byte

offset is a numeric expression returning an integer in the range 0 to 65535 and indicates the address of the memory location where the data is to be written. It is the offset from the current segment, which was set by the last DEF SEG

statement.

byte is the data byte. It must be in the range 0 to 255.

Remarks POKE can pass arguments to assembly

language routines.

If either offset or byte is outside the specified range, an "Illegal function call" error is

returned.

PEEK is the complementary function to

POKE.

Example 10 POKE &H5A00,&HFF

Warning Use POKE carefully. If it is used incorrectly, it

can cause GWBASIC or MS-DOS to crash.

Returns the current horizontal (column)

position of the cursor.

Syntax POS (dummy)

dummy is a dummy argument. Any value is accepted.

Remarks The current horizontal (column) position of the

cursor is returned. The leftmost position is 1. The rightmost position may be 40 or 80, depending on the current screen width. See

CSRLIN and LPOS Functions.

Example IF POS(0) > 50 THEN BEEP

Draws a point at the specified position on the

Syntax

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

screen (Graphics Mode only).

x,y

If the STEP option is not included, x,y are absolute coordinates of the point to be drawn. If the STEP option is included, x,y are the relative coordinates of the point to be drawn.

color

is the color number to be used, in the range 0 to 3. (See the COLOR graphics statement for the current screen mode, for details.) If no 'color' parameter is given, the graphics background color is selected.

Remarks

If the color is given, PRESET is identical to PSET. If an out of range coordinate is given, no action is taken and no error message is given. A color greater than 3 results in an "Illegal function call".

Examples

PRESET (x,y)

is identical to:

PSET (x,y),0

assuming that the graphics background color is 0 (Black). See the COLOR graphics statement for the current mode.

Outputs data to the screen.

Syntax

PRINT [list-of-expressions {, |; }]

list-of-expressions

the expressions in the list may be numeric and/or string expressions. (String constants must be enclosed in quotation marks.) Each expression should be separated from the next by a comma, blank, or semicolon.

Remarks

If you include the "list-of-expressions", they are displayed on the screen. If you omit the "list-of-expressions", a blank line is displayed. A question mark may be used in place of the word PRINT in a PRINT statement.

The position of each printed item is determined by the punctuation used to separate the items in the list. GWBASIC divides the line into print zones of 14 spaces each. In the list of expressions, a comma causes the next value to be printed at the beginning of the next zone. A semicolon causes the next value to be printed immediately after the last value. One or more spaces between expressions has the same effect as typing a semicolon.

If a comma or a semicolon terminates the list of expressions, the next PRINT or PRINT USING statement begins printing on the same line spacing accordingly. If the list of expressions terminates without a comma or a semicolon, a carriage return is

printed at the end of the line. If the printed line is longer than the terminal width, GW BASIC goes to the next physical line and continues printing.

Printed numbers are always followed by a space. Positive numbers are preceded by a space. Negative numbers are preceded by a minus sign. Single precision numbers that can be represented with 7 or fewer digits in the unscaled format are output using the unscaled format. For example, 10-7 is output as .0000001 and 10-8 is output as 1E-08. Double precision numbers that can be represented with 16 or fewer digits in the unscaled format are output using the unscaled format. For example, 1D-15 is output as .0000000000000000001 and 1D-16 is output as 1D-16.

Examples

5 REM PRINT WITH COMMAS 10 X = 5 20 PRINT X + 5,X-5,X*(-5),X %5 30 END RUN 10 0 -25 3125 Ok

```
5 REM WITH SEMICOLON AT 20
10 INPUT X
20 PRINT X "SQUARED IS" X^2 "AND";
30 PRINT X "CUBED IS" X^3
40 PRINT 'BLANK LINE
RUN
? 9
```

9 SQUARED IS 81 AND 9 CUBED IS 729

Ok

```
5 REM NUMBERS WITH SEMICOLONS
10 FOR X = 1 TO 5
20 J = J + 5
30 K = K + 10
40 ?J;K;
50 NEXT X
RUN
5 10 10 20 15 30 20 40 25 50
Ok
```

Outputs data to the screen using a specified format.

Syntax PRINT USING format string;

list of expressions{,|;}

format string is a string expression composed of special formatting characters.

These formatting characters (see below) determine the field and the format of the printed strings or numbers.

list of is comprised of the string expressions or numeric expressions that are to be printed, separated by semicolons, or commas. expressions String constants must be enclosed in quotation marks. If a comma or a semicolon terminates the list of expressions, the next PRINT or PRINT USING statement begins printing on the same line, spacing accordingly.

> When PRINT USING is used to print strings, one of three formatting characters may be used to format the string field:

Specifies that only the first character in the given string is to be printed.

Specifies a number of characters to be printed. If two backslashes are typed with no spaces, two characters will be printed; with one space, three characters will be printed, and so on. If the string is longer than the field, the extra characters are ignored.

//

&

If the field is longer than the string, the string will be left-justified in the field and padded with spaces on the right. For example:

10 AS="LOOK":BS="OUT"
30 PRINT USING "!";AS;BS
40 PRINT USING "\ \";AS;BS
50 PRINT USING "\ \";AS;BS;"!!"
RUN
LO
LOOKOUT
LOOK OUT!!

Specifies a variable length string field. When the field is specified with "&", the string is output without modification. For example:

10 A\$="LOOK":B\$="OUT"
20 PRINT USING "!";A\$;
30 PRINT USING "&";B\$
RUN
LOUT

#

When PRINT USING is used to print numbers, the formatting special characters may be used to format the numeric field:

Represents each digit position. Digit positions are always filled. If the number to be printed has fewer digits than positions specified, the number will be right-justified (preceded by spaces) in the field.

A decimal point can be inserted at any position in the field. If the format string specifies that a digit is to precede the decimal point, the digit will always be printed (as 0, if necessary). Numbers are rounded as necessary. For example:

PRINT USING "##.##";.78
0.78
PRINT USING "##.##";987.654
987.65
PRINT USING "##.##" ";10.2,5.3,66.789,.234
10.20 5.30 66.79 0.23

In the last example, three spaces were inserted at the end of the format string to separate the printed values on the line.

A plus sign at the beginning or end of the format string will cause the sign of the number (plus or minus) to be printed before or after the number.

A minus sign at the end of the format field will cause negative numbers to be printed with a trailing minus sign. For example:

PRINT USING "+##.## ";-68.95,2.4,55.6,-.9 -68.95+2.40+55.60-0.90 PRINT USING "##.##-";-68.95,22.449,-7.01 68.95-22.45 7.01-

A double asterisk at the beginning of the format string causes leading spaces in the numeric field to be filled with asterisks. The ** also specifies positions for two more digits. For example:

PRINT USING "**#.#";12.39,-0.9,765.1 *12.4*-0.9765.1

A double dollar sign causes a dollar sign to be printed to the immediate left of the formatted number. The \$\$ specifies two more digit positions, one of which is the dollar sign. The exponential format cannot be used with \$\$. Negative numbers cannot be used unless the minus sign trails to the right. For example:

PRINT USING "\$\$###.##";456.78 \$456.78

* *

**\$

The **\$ at the beginning of a format string combines the effects of the above two symbols. Leading spaces will be asterisk-filled and a dollar sign will be printed before the number.
**\$ specifies three more digit positions, one of which is the dollar sign. For example:

PRINT USING "***\$##.##";2.34 ***\$2.34

A comma that is to the left of the decimal point in a formatting string causes a comma to be printed to the left of every third digit to the left of the decimal point. A comma that is at the end of the format string is printed as part of the string. A comma specifies the digit position for itself. The comma has no effect if used with the exponential (^^^) format. For example:

PRINT USING "####,.##";1234.5 1,234.50 PRINT USING "####.##,";1234.5 1234.50.

Four carets (or up-arrows) may be placed after the digit position characters to specify exponential format. The four carets allow space for E+xx or D+xx to be printed. Any decimal point position may be specified. The significant digits are left-justified, and the exponent is adjusted. Unless a leading+or trailing + or - is specified, one digit position will be used to the left of the decimal point to print a space or a minus sign. For example:

2000

PRINT USING "##.##^^^";234.56 2.35E+02 PRINT USING ".####^^^- ";888888 .8889E+06 PRINT USING "+.##^^^";123 +.12E+03

An underscore in the format string causes the next character to be output as a literal character. For example:

PRINT USING "_! ##.## _!";12.34 !12.34!

The literal character itself may be an underscore by placing "__" in the format string.

If the number to be printed is larger than the specified numeric field, a percent sign is printed in front of the number. If rounding causes the number to exceed the field, a percent sign will be printed in front of the rounded number. For example:

PRINT USING "##.##";111.22 %111.22 PRINT USING ".##";.999 %1.00

%

If the number of digits specified exceeds 24, an "Illegal function call" error will result.

PRINT# and PRINT# USING

Statements

Write data sequentially to a disk file. PRINT# and PRINT# USING are usually used in a program.

DD IN

Syntax PRINT# filenum, [USING format-string

;] list-of-expressions

filenum is the number used when the file was OPENed for

OUTPUT

format-string is a string expression (usually a constant or variable)

composed of formatting characters described in the "PRINT

USING" statement

list-of-expressions

is a list of the numeric and/or string expressions to be

written to file

PRINT# does not compress data on the disk. An image of the data is written to the disk, just as it would be displayed on the terminal screen with a PRINT statement. Be sure to delimit the data so that it can be input correctly.

Numeric expressions should be delimited by semicolons.

Example 50 PRINT#1,B;C;D;X;Y;Z

If commas are used as delimiters, the extra blanks that are inserted between print fields will also be written to the disk. String expressions must be separated by semicolons in the list. To format the string expressions correctly on the disk, use explicit delimiters in the list of expressions.

Example

Let A\$="CAMERA" and B\$='93604-1'. The statement:

100 PRINT#1,A\$;B\$

writes CAMERA93604-1 to the disk. Because there are no delimiters, this could not be input as two separate strings. Insert explicit delimiters into the PRINT# statement as follows:

200 PRINT#1,A\$;",";B\$

The image written to disk is

CAMERA,93604-1

If the strings themselves contain commas, semicolons, significant leading blanks, carriage returns, or line feeds, write them to disk surrounded by explicit quotation marks, CHR\$(34).

Example

100 A\$ = "CAMERA, AUTOMATIC" 200 B\$ = "93604-1" 300 PRINT#1,A\$;B\$

Writes the following image to disk:

CAMERA, AUTOMATIC 93604-1

The statement

400 INPUT#1,A\$,B\$

Inputs "CAMERA" to A\$ and "AUTOMATIC 93604-1" to B\$. To separate these strings properly on the disk, write double quotation marks to the disk image using CHR\$(34). The statement:

500 PRINT#1,CHR\$(34);A\$;CHR\$(34); CHR\$(34);B\$;CHR\$(34)

writes:

"CAMERA, AUTOMATIC"" 93604-1"

And the statement:

600 INPUT#1,A\$,B\$

Inputs "CAMERA, AUTOMATIC" to A\$ and "93604-1" to B\$.

The PRINT# statement may also be used with the USING option to control the format of the disk file:

700 PRINT#1,USING"\$\$###.##,";J;K;L

See Chapter 4 (Disk File Handling) and "WRITE#" in this chapter.

Illuminates a pixel at a specified position on the screen. (Graphics Mode only.)

Syntax

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

x,y

If the STEP option is not included, x,y are absolute coordinates of the point to be drawn. If the STEP option is included, x,y are the relative coordinates of the point to be drawn.

color

is the color number to be used, in the range 0 to 3. (See the COLOR graphics statement for the current screen mode, for details.) If no "color" parameter is given, the graphics foreground color is selected.

Remarks

PSET differs from PRESET in the default if no color is specified. PSET defaults to the foreground color. PRESET defaults to the background color.

See PRESET.

Examples:

5 REM DIAGONAL LINE 10 FOR i=0 TO 100 20 PSET (i,i) 30 NEXT

10 REM CLEARS OUT LINE BY SETTING EACH PIXEL TO 0:
40 FOR i = 100 TO 0 STEP -1
50 PSET (i,i),0
60 NEXT

Writes a specified number of bytes to a communications file.

Syntax PUT [#] filenum [,length]

filenum is an integer expression returning a valid file number

length is an integer expression returning the number of bytes to be transferred out of the communications buffer. 'length'

cannot exceed the value set by the /S: switch when GWBASIC was invoked or the value optionally set in the OPEN

statement.

Example 100 PUT #2, 80

Remarks This is ordinarily used only in a program, not

in direct mode.

Writes a record from a random buffer to a random file.

Syntax PUT [#]filenum [, recordnum]

filenum is the number under which the file was OPENed

recordnum specifies the number of the record in the file. It must be in the range 1 to 32,767. If omitted the current record number is assumed (i.e., the record whose number is one

higher than that of the last record accessed).

PUT (FILES) is usually used in a program, Remarks

not in direct mode. PRINT#. PRINT# USING. WRITE#, LSET and RSET may be used to put

characters in the random file buffer before

executing a PUT statement.

In the case of WRITE#, GWBASIC pads the

buffer with spaces up to the carriage return.

Possible Any attempt to read or write past the end of

the buffer causes a "Field overflow" error. Errors

Example

```
10 OPEN "R",1,"A:RAND",48
20 FIELD 1.20 AS R1$.20 AS R2$.8 AS R3$
30 FOR L=1 TO 2
40 INPUT "name"; N$
50 INPUT "address":M$
60 INPUT "phone";P#
70 LSET R1$ = N$
80 LSET R2$ = M$
90 LSET R3$ = MKS$(P#)
100 PUT #1,L
110 NEXT L
120 CLOSE #1
130 END
RUN
name? Super man
address? usa
phone? 11234621
name? robin hood
address? England
phone? 23462101
Ok
```

Statement 10 opens the random file RAND, with a record length of 48 on the diskette drive in A. The file number is 1. Statement 20 divides the buffer into fields.

Statement 100 writes a record to file RAND, with the record number being set by the control variable of the FOR/NEXT loop.

Transfers the graphics image stored in an array to the screen.

PUT (x,y), array [,action_verb] Syntax

represent the top left corner of the rectangle to be displayed.

is the name of an array containing the image to be displayed. array The type of the array must be numeric.

is one of: PSET, PRESET, AND, OR, XOR, The default 'action action verb verb' is XOR.

> The PUT and GET statements are used to transfer graphics images to and from the screen. PUT and GET make possible animation and high-speed object motion in graphics mode.

The array is used simply as a place to hold the image and can be of any type except string. It must be given dimensions large enough to hold the entire image.

The PUT statement transfers the image stored in the array onto the screen. The specified point is the coordinate of the top left corner of the image.

x,y

Remarks

The Action Verb Parameter

The "actionverb" specifies the interaction between the stored image and the one already on the screen.

PSET transfers the data point by point onto the screen. Each point has the exact color taken from the screen with a GET.

PRESET is the same as PSET except that a negative image is produced.

AND transfers the data over an existing image on the screen. The resulting image is the product of the logical AND expression. Points that had the same color in both the existing image and the PUT image will remain the same color, in original, but should be points that do not have the same color in both images will be changed.

OR superimposes the image onto an existing image.

XOR causes the points on the screen to be INVERTED where a point exists in the array image. This behavior is exactly like that of the cursor. When an image is PUT against a complex background TWICE, the background is restored unchanged. This allows you to move an object around the screen without erasing the background.

In medium resolution AND, OR and XOR have the following effects on color:

AND					OR							
		SC	ree	en				sc	re	en		
a r		0	1	2	3	a r		0	1	2	3	
r	П					r						
a	0	0	0	0	0	a	0	0	1	2	3	
У						у						
	1	0	1	0	1		1	1	1	3	3	
v						v						
a	2	0	0	2	2	a	2	2	3	2	3	
l		ı				l						
u	3	0	1	2	3	u	3	3	3	3	3	
e						e						E

	XOR screen								
a		0	1	2	3				
r	Н	H	_	_	_				
r		_	1	0	0				
a	0	0	1	2	3				
У	1	1	0	3	2				
v	П								
a	2	2	3	0	1				
l			^		_				
u	3	3	2	1	0				
0									

Animation

Animation proceeds as follows:

- PUT the object(s) on the screen (with the XOR option)
- Recalculate the new position of the object(s)
- PUT the object(s) on the screen (with the XOR option) a second time at the old location(s) to remove the old image(s)
- Go to step 1 for the new location.

PUT (Graphics) Statement

Movement done this way will leave the background unchanged. Minimize the time between steps 4 and 1, and make sure that there is enough time delay between 1 and 3 to eliminate flickering images. If more than one object is being animated, every object should be processed at once, one step at a time.

PSET can perform faster animations, but will not preserve the background. This method must use an image large or larger than the maximum distance the object will move. Thus, when an object is moved, this border will effectively erase any points left by the previous PUT. This may be faster than the method using XOR described above, since only one PUT is required to move an object (although you must PUT a larger image).

Possible Errors

An "Illegal function call" error occurs if the image is too large to fit on the screen. Reseeds the random number generator.

Syntax

RANDOMIZE [numexp]

numexp

is any numeric expression. The value of the expression will be used to seed the random numbers.

Remarks

If "numexp" is omitted, GWBASIC suspends program execution and asks for a value by displaying:

Random Number Seed (-32768 to 32767)?

before executing RANDOMIZE.

To get a new random seed without prompting the user, use the numeric TIMER function. For example:

RANDOMIZE TIMER

If the random number generator is not reseeded, the RND function returns the same sequence of random numbers each time the program is RUN. To change the sequence of random numbers every time the program is RUN, place a RANDOMIZE statement at the beginning of the program and change the argument with each RUN.

RANDOMIZE Statement

Example

10 RANDOMIZE 20 FOR I=1 TO 3 **30 PRINT RND;** 40 NEXT I Ok RUN Random Number Seed[-32768to32767]?3 .2226007 .3343962 .7341223 Ok RUN Random Number Seed [-32768 to 32767]?4 .9468615 .5775203 .6716166 Ok RUN Random Number Seed[-32768 to 32767]?3 .2226007 .3343962 .7341223

Reads values from one or more DATA statements and assigns them to variables.

Syntax

READ variable [, variable]...

variable

each variable in the list may be a numeric or string variable. The type of the variable must agree with the type of the associated value in the DATA sequence.

Remarks

A READ statement must always be used in conjunction with a DATA statement. READ statements assign variables to DATA statement values on a one-to-one basis. If the data type (numeric or string) of an entry in the data sequence does not correspond to the type of the associated variable, a "Syntax error" will result. However any numeric data type (integer, single or double precision) may be assigned to any numeric variable.

A single READ statement may access one or more DATA statements (they will be accessed in order), or several READ statements may access the same DATA statement. If the number of variables in the list of variables exceeds the number of elements in the DATA statement(s), an "Out of data" error message is printed. If the number of variables specified is fewer than the number of elements in the DATA statement(s), subsequent READ statements will begin reading data at the first unread element. If there are no subsequent READ statements, the extra data is ignored.

To reread DATA statements from the start, use the RESTORE statement (see "RESTORE" later in this chapter).

Example 1

80 FOR I = 1 TO 10 90 READ A(I) 100 NEXT I 110 DATA 3.08,5.19,3.12,3.98,4.24 120 DATA 5.08,5.55,4.00,3.16,3.37

This program segment READs the values from the DATA statements into the array A. After execution, the value of A(1) will be 3.08, and so on.

Example 2

10 PRINT "CITY", "STATE", "ZIP"
20 READ C\$,S\$,Z
30 DATA "DENVER,", COLORADO, 80211
40 PRINT C\$,S\$,Z
Ok
RUN
CITY STATE ZIP
DENVER, COLORADO 80211
Ok

This program READs string and numeric data from the DATA statement in line 30.

Allows explanatory remarks to be inserted in a program.

Syntax

REM remark

remark

represents a sequence of characters

Remarks

REM statements are not executed but are output exactly as entered when the program is listed.

REM statements may be branched into from a GOTO or GOSUB statement. Execution will continue with the first executable statement after the REM statement.

Remarks may be added to the end of a line by preceding the remark with a single quotation mark (') instead of REM. The single quotation mark may also be entered just after the line number, like REM.

Note

Do not use remarks in a DATA statement, because they would be considered legal data.

Example

120 REM Calculate Average Velocity 130 FOR I = 1 TO 20 140 SUM = SUM + V(I) 150 NEXT I 160 AV = SUM/20

or

120 FOR I = 1 TO 20 'Calculate 125 'Average Velocity 130 SUM = SUM + V(I) 140 NEXT I 150 AV = SUM/20

or

120 'Calculate Average Velocity 130 FOR I=1 TO 20 140 SUM=SUM+V(I) 150 NEXT I 160 AV=SUM/20 Changes the line numbers of the current program.

Syntax

RENUM [new linenum] [, [old linenum] [, increment]]

new linenum

is the first line number to be used in the new sequence. The default is 10.

old linenum

is the line in the current program where renumbering is to begin. The default is the first line of the program.

increment

is the increment to be used in the new sequence. The default is 10.

RENUM also changes all line number references following GOTO, GOSUB, THEN, ON . . . GOTO, ON . . . GOSUB, RESTORE, RESUME, and ERL statements to reflect the new line numbers. If a nonexistent line number appears after one of these statements, the error message "Undefined line number xxxxx in yyyyy" is printed. The incorrect line number reference is not changed by RENUM, but line number yyyyy may be changed.

Note

RENUM cannot be used to change the order of program lines (for example, RENUM 15,30 when the program has three lines numbered 10, 20 and 30) or to create line numbers greater than 65529. An "Illegal function call" error will result.

Examples RENUM

Renumbers the entire program. The first new line number will be 10. Lines will be numbered in increments of 10.

RENUM 300,,50

Renumbers the entire program. The first new line number will be 300. Lines will be numbered in increments of 50.

RENUM 1000,900,20

Renumbers the lines from 900 up, so they start with line number 1000 and are numbered in increments of 20.

Closes all open data files on all drives.

Syntax

RESET

Remarks

RESET closes all open data files on all drives, and forces all blocks in memory to be written to disk. Thus, if the machine loses power, all files will be properly updated. All files must be closed before a disk is removed from its drive.

Note that RESET performs the same action as CLOSE with no arguments, if all open data files are residing on disk.

Permits DATA statements to be re-read either from the beginning of the internal data file or from a specified line.

Syntax

RESTORE [linenum]

linenum

must be the line number of a DATA statement

Remarks

After a RESTORE statement is executed, the next READ statement accesses the first item in the first DATA statement in the program. If "linenum" is specified, the next READ statement accesses the first data item in the specified DATA statement.

Example

10 READ A, B, C 20 RESTORE 30 READ D, E, F 40 DATA 58, 67, 97 50 PRINT A; B; C; D; E; F RUN 58 67 97 58 67 97 Ok Continues program execution after an error trapping routine has been performed.

Syntax

RESUME { 0 | NEXT | linenum}

RESUME

execution resumes at the statement which caused the error

RESUME 0

execution resumes at the statement immediately following the one which caused the error

RESUME linenum

NEXT

execution resumes at the specified line

Remarks

Any one of the four formats shown above may be used, depending upon where execution is to resume.

A RESUME statement that is not in an error handling routine causes a "RESUME without error" message to be printed.

Example

10 ON ERROR GOTO 900

.

900 IF (ERR = 230) AND (ERL = 90) THEN 910 PRINT "TRY AGAIN" : RESUME 80 Returns a substring from a specified string, extracting its rightmost characters.

RIGHT\$ (string, length) Syntax

string is a string expression whose value is the original string from

which a substring is to be returned

length is a numeric expression rounded to the nearest integer,

whose value (from 0 to 255) represents the length of the

returned string

Remarks If "length" is greater or equal to LEN(string),

> then the entire original string is returned. When length=0, the null string (length of zero)

is returned.

10 A\$ = "THIS IS GWBASIC" Example

20 PRINT RIGHTS(AS,5)

RUN BASIC Ok

Also see the LEFT\$ and MID\$ functions in

this chapter.

Removes an existing directory.

RMDIR pathname Syntax

is the name of the directory which is to be deleted pathname

RMDIR works exactly like the MS-DOS Remarks command RMDIR. The directory to be deleted must be empty of all files and sub-directories except the working directory ('.') and the parent directory ('..') entries, or a "Path not

found" error is given.

7-287

With reference to our sample structure above, we decide that we no longer want the sub-directory ANDY. Let us assume that our current directory is ROOT. Then:

RMDIR "ACCOUNTING\ANDY"

deletes the directory ANDY.

On the other hand, if you want to make ACCOUNTING the current directory and remove the directory called AMOS then:

CHDIR "ACCOUNTING" RMDIR "AMOS"

Possible Errors

"Bad File name"

"Path/File Access error" usually indicates that the directory is not empty.

Returns a random number between 0 and 1.

Syntax

RND [numexp]

numexp

is a numeric expression which affects the returned value. See the following "Remarks" section.

Remarks

The same sequence of random numbers is generated each time the program is RUN unless the random number generator is reseeded. You may reseed the generator either by the RANDOMIZE statement or by the RND function (specifying numexp<0). numexp<0 always restarts the same sequence for any given "numexp". This sequence is not affected by RANDOMIZE, so if you want to generate a different sequence each time the program is run, you have to use a different value of numexp each time.

If numexp>0 or is omitted, RND(numexp) generates the next random number in the sequence. RND(0) repeats the last number generated.

To get integer random number in the range 0 (zero) to N, use:

INT (RND*(N+1))

RND Function

Example

10 FOR I = 1 TO 5 20 PRINT INT(RND*100); 30 NEXT RUN 24 30 31 51 5 Ok Runs the current program or loads a program from disk and runs it.

Syntax 1 RUN [linenum]

Syntax 2 RUN filename [,R]

linenum is the line number of the program resident in memory where

the execution must begin

filename is a string expression which specifies the program to be

loaded and run

R if this option is specified all data files (that were opened

before loading the designated program) remain open

Remarks

If "linenum" is specified, execution begins on that line. Otherwise, execution begins at the lowest line number. GWBASIC always returns to command level after a RUN command. The name used when the file was SAVEd is the name specified by "filename" or "pathname". (MS-DOS will append a default .BAS filename extension if one was not supplied in the SAVE command.) RUN {filename} closes all open files and deletes the current contents of memory before loading the designated program. However, with the "R" option, all data files remain open.

RUN "B:NEWFILE", R RUN A\$

RUN 150

RUN "C:\R001\R002"

Saves the current program on disk and gives it a name. Option A saves the program in ASCII format. Option P saves it protected.

Syntax

SAVE filename [, { A | P }]

filename

is a string expression which specifies the name of the file to be saved, and optionally the drive. If the filename extension is omitted, .BAS is assumed. If the drive is omitted, the default MS-DOS drive is assumed.

Α

the A option will save the program in ASCII format. Otherwise GWBASIC saves the file in a compressed binary format. Programs saved in ASCII may be read as Data Files or MERGEd.

P

the P option will save the program in an encoded binary format. This is the protection option. When a protected program is later RUN (or LOADed), any attempt to LIST or EDIT it will fail with an "Illegal function call" error. No way is provided to "unprotect" such a program.

Remarks

If a file with the same name already exists on the selected disk, it will be written over.

ASCII format takes more space on disk, but some disk access requires that files be in ASCII format. Attempts to MERGE binary programs will result in a "Bad File Mode" error.

If the filename is eight characters or less and no extension is supplied, MS-DOS adds the extension .BAS to the name.

Examples

SAVE "SUPERB"

Saves the program in memory on the default drive as SUPERB.BAS.

SAVE "A:PROG",A

Saves the program in memory in ASCII form on the diskette inserted on drive A, as PROG.BAS; it may be later MERGEDd.

SAVE "B:SECRET",P

Saves protected the program in memory on the diskette inserted on drive B, in protected form as SECRET.BAS; it may not be altered.

Returns the ASCII code (0-255) or the color number for the character at the specified row and column.

Syntax

SCREEN (row, column [,condition])

row

is a numeric expression returning an unsigned integer in the range 1 to 25

column

is a numeric expression returning an unsigned integer in the range 1 to 40 or 1 to 80 depending on the width

condition

is a valid numeric, relational or logical expression returning a boolean result (0 or 1). condition is only valid in Text Mode.

Remarks

In text mode, if **condition** is given as non-zero, the color number for the character is returned instead of the ASCII code. The color is a number in the range 0-255. This number (x) may be interpreted as follows:

- (x MOD 16) is the foreground color
- (((x-foreground)/16)MOD 128) is the background color
- (x>127) is true (-1) if the character is blinking, false (0) if not.

Refer to Appendix A for a complete list of ASCII codes. The colors associated with each number are listed under the COLOR Command.

Remarks

In graphics mode the SCREEN function returns zero if the specified location contains graphics information.

Examples

100 X = SCREEN (10,10)

If the character at 10,10 is A, then x will contain 65.

110 X = SCREEN (1,1,1)

Returns the color number of the character in the upper left hand corner of the screen.

Sets the screen attributes that will be used by subsequent statements.

Syntax

SCREEN [mode] [, [burst] [, [apage] [,vpage]]]

mode

is a numeric expression resulting in an integer value of 0, 1, 2, or 100. It defines either Text Mode (0), Medium-Resolution Graphics Mode (1), High-Resolution Graphics Mode (2), or Super-Resolution Graphics Mode (100).

burst

is a numeric expression resulting in an integer value of 0 or 1. It enables color on a color TV set. In Text Mode (mode \pm 0) a 0 value disables color, and a 1 value enables color. In Medium-Resolution Graphics Mode (mode \pm 1) a 0 value enables color, and a 1 value disables color. Both in High-Resolution and Super-Resolution Graphics Mode (mode \pm 2 or 100) the burst value is ignored, as these two modes only support monochrome. For a standard monitor, this parameter has no meaning.

apage

is an integer expression in the range 0 to 7 for width 40, or 0 to 3 for width 80. It selects the active page, i.e. the page to be written to by output statements to the screen. If omitted, the active page defaults to 0. This parameter is valid in Text Mode only.

vpage

is an integer expression in the range 0 to 7 for width 40, or 0 to 3 for width 80. It selects the visual page, i.e. the page to be displayed on the screen. The visual page may be different from the active page. If omitted, the visual page defaults to the active page. This parameter is valid in Text Mode only.

Mode and Burst Parameters

In the following table the first two columns are the "mode" and "burst" parameters of a SCREEN statement.

Mode	Burst	Resolution
0	0	80 c. x 25 r B/W
		Text Mode
0	1	80 c. x 25 r Color
		Text Mode
1	0	320 hor.pixels x 200 vert.
		pixels-
		Color Medium Resolution
		Graphics
		(40 c. x 25 r.)
1	1	320 hor.pixels x 200 vert.
		pixels-
		B/W Medium Resolution
		Graphics
		(40 c. x 25 r.)
2	x	640 hor.pixels x 200 vert.
		pixels-
		B/W High Resolution
		Graphics
		(80 c. x 25 r.)
100	x	640 hor.pixels x 400 vert.
		pixels-
		B/W Super Resolution
		Graphics
		(80 c. x 25 r.)

Default Values

If you do not enter a SCREEN statement, the system assumes the following default values:

```
mode = 0 (Text Mode)
burst = 0 (B/W)
apage = 0 (active page 0)
vpage = 0 (virtual page 0)
```

It would be the same as if you entered:

SCREEN 0,0,0,0

Apage and Vpage Parameters

If Text Mode is selected, you can specify two more parameters "apage" and "vpage" to select the active and visual page. There are eight display pages (numbered 0 to 7) in 40-column Text Mode, and four display pages (numbered 0 to 3) in 80-column Text Mode. Only one display page is available in any of the three graphics modes.

Screen Width

At initialization the width is 80 columns, thus you should use the WIDTH statement if you want to select a 40-column screen. If you select the medium resolution mode by the SCREEN statement, this also causes the number of columns to be 40 (without using the WIDTH statement).

While in Text Mode, the WIDTH statement may be used to select between the 40-column mode and the 80-column mode. Likewise, the WIDTH statement may be used to select between modes 1 and 2 (medium or high-resolution mode). See the WIDTH statement in this chapter.

If all parameters are valid the new screen mode is stored, the screen is erased, the foreground and the background colors are set to their default values. The SCREEN statement must precede any I/O statement to the screen, but you can use more than one SCREEN statement to define different screen attributes for different sections of your program.

If all parameters are identical to the preceding ones nothing is changed or erased.

If you omit a parameter, it keeps the old value (except that the visual page defaults to the active page.)

If you are in Text Mode and you switch active pages back and forth, you should save the cursor position on the current active page (see POS(0) and CSRLIN) before changing to another active page. #Note: There is only one cursor shared among all the pages.

If you are in Text Mode and you return to the original page you can restore the cursor position by the LOCATE (Text) statement.

Examples

100 SCREEN 0,1,1,2

Selects 80-column text-mode with color, sets active page to 1 and visual page to 2.

200 SCREEN 1,0

Switches to 40-column medium-resolution color graphics.

300 SCREEN 0

Switches back to text-mode. The omitted parameters assume the old values (except the visual page that defaults to the active page). Note that, if the last SCREEN statement executed was statement 200, then statement 300 would switch to 40-column BW text mode and set the active and visual pages to 0.

SGN Function

Returns 1 if the argument is positive, 0 if the argument is zero, and -1 if the argument is negative.

Syntax SGN (numexp)

Remarks If numexp>0, SGN(numexp) returns 1.

If numexp+0, SGN(numexp) returns 0. If numexp<0, SGN(numexp) returns -1.

Example 50 ON SGN (X)+2 GOTO 300,400,500

branches to:

300 if numexp is negative, 400 if numexp is 0, and 500 if numexp is positive.

Calculates the sine of the argument.

Syntax SIN(numexp)

numexp is a numeric expression representing the angle in radians.

Remarks The SIN function is calculated in single

precision, unless "'/D" is supplied in the

GWBASIC command line.

Example PRINT SIN (1.5)

See also the COS function in this chapter.



Produces sound via a speaker.

Syntax SOUND frequency, duration

frequency is a numeric expression in the range 37 to 32767. It represents

the Hertz frequency (cycles per second).

duration is the duration in clock ticks. Clock ticks occur 18.2 times per

second. Duration is an integer expression in the range 0 to 65535.

Example 100 SOUND RND*100+37,2

This statement creates random sounds.

Remarks Sound x,0 or SOUND 32767,x cause silence.

Notes and Frequencies

The following table correlates notes with their frequencies for two octaves.

Note	Frequency	Note	Frequency
C	130.810	C*	523.250
D	146.830	D	587.330
\mathbf{E}	164.810	${f E}$	659.260
F	174.610	\mathbf{F}	698.460
G	196.000	G	783.990
Α	220.000	Α	880.000
В	246.940	В	97.770
C	261.630	C	1046.500
D	293.660	D	1174.700
\mathbf{E}	329.630	${f E}$	1318.500
F	349.230	\mathbf{F}	1396.900
G	392.000	G	1568.000
Α	440.000	Α	1760.000
В	493.880	В	1975.500

*middle C

Doubling a frequency approximates a note one octave higher. Halving it approximates a note one octave lower.

Tempos and Beats/Minute

This table shows typical tempos in terms of clock ticks.

Tempo		Beats/	Ticks/
-		Minute	Beat
very slow	Larghissimo		
	Largo	40-60	28.13-18.75
	Larghetto	60-66	18.75-17.05
	Grave		
	Lento		
	Adagio	66-76	17.05-14.8
slow	Adagietto		
	Andante	76-108	14.8-10.42
medium	Andantino		
	Moderato	108-120	10.42-9.38
fast	Allegretto		
	Allegro	120-168	9.38-6.7
	Vivace		
	Veloce		
	Presto	168-208	6.7-5.41
very fast	Prestissimo		

Returns a string of a specified number of spaces.

Syntax SPACE\$ (length)

length is an integer expression in the range 0 to 255. It specifies the number of spaces i.e. the length of the returned string.

Example

```
10 FOR I + 1 TO 5
20 X$ + SPACE$[I]
30 PRINT X$;I
40 NEXT I
RUN
1
2
3
4
5
```

Skips "n" spaces in a PRINT, LPRINT, or PRINT# statement.

Syntax

SPC (n)

n

is an integer expression in the range 0 to 255. It specifies the number of spaces to be inserted in the output line.

Remarks

SPC may only be used with PRINT, LPRINT and PRINT# statements.

If "n" is greater than the defined width, then the value used is "n MOD width". A semicolon (;) is assumed to follow the SPC function; thus GWBASIC does not add a carriage return if the SPC function is at the end of the list of data items.

Example

PRINT "FOUR" SPC(15) "SEASONS" FOUR SEASONS

See also the SPACE\$ function in this chapter.

Returns the square root of a positive numeric expression.

Syntax

SQR (numexp)

Remarks

SQR is calculated in single precision, unless "/D" is supplied in the GWBASIC command line.

An "Illegal function call" error results if the argument is negative.

Example

10 FOR X = 10 TO 25 STEP 5 20 PRINT X, SGR(X) 30 NEXT X RUN 10 3.162278 15 3.872984 20 4.472136 25 5

STICK

Function

Returns the x and y coordinates of two joysticks.

Syntax

v = STICK(n)

n

is a numeric expression in the range 0 to 3 which affects the result as follows:

- 0 returns the x coordinate for joystick A.
- 1 returns the y coordinate of joystick A.
- 2 returns the x coordinate of joystick B.
- 3 returns the y coordinate of joystick B.

Note: STICK(0) retrieves all four values for the coordinates, and returns the value for STICK(0). STICK(1), STICK(2), and STICK(3) do not sample the joystick. They get the values previously retrieved by STICK(0).

Remarks

The range of values for x and y depends on your particular joysticks.

Example:

10 PRINT "Joystick B"
20 PRINT "x","y"
30 FOR J = 1 TO 100
40 TEMP = STICK(0)
50 X = STICK(2): Y = STICK(3)
60 PRINT X,Y
70 NEXT

This program takes 100 samples of the coordinates of joystick B and prints them.

Terminates program execution and returns to command level. STOP is only used in a program.

Syntax

STOP

Ok

Remarks

A STOP statement may be used anywhere in a program. When a STOP is encountered, the following message is displayed:

Break in nnnnn

Where nnnn is the line number containing the STOP statement.

The STOP statement does not close files, unlike the END statement.

GWBASIC always returns to command level after a STOP is executed. The CONT command resumes execution.

Example

10 INPUT A,B,C
20 K = A^2*5.3:L = B^3/.26
30 STOP
40 M = C*K + 100:PRINT M
RUN
?1,2,3
BREAK IN 30
Ok
PRINT L
30.76923
Ok
CONT
115.9

Returns the status of the joystick buttons (triggers).

Syntax

As a statement:

STRIG ON

STRIG OFF

As a function:

v = STRIG(n)

n

is a numeric expression in the range 0 to 7. It affects the value returned by the function as follows:

- 0 Returns -1 if button A1 was pressed since the last STRIG(0) function call, returns 0 if not.
- 1 Returns -1 if button A1 is currently pressed, returns 0 if not.
- 2 Returns -1 if button B1 was pressed since the last STRIG(2) function call, returns 0 if not.
- 3 Returns -1 if button B1 is currently pressed, returns 0 if not.

- 4 Returns -1 if button A2 was pressed since the last STRIG(4) function call, returns 0 if not.
- 5 Returns -1 if button A2 is currently pressed, returns 0 if not.
- 6 Returns -1 if button B2 was pressed since the last STRIG(6) function call, returns 0 if not.
- 7 Returns -1 if button B2 is currently pressed, returns 0 if not.

Remarks

STRIG ON must be executed before any STRIG(n) function calls may be made. After STRIG ON, every time the program starts a new statement BASIC checks to see if a button has been pressed.

If STRIG is OFF, no testing takes place.

Enables and disables trapping of the joystick buttons.

Syntax

STRIG(n) ON

STRIG(n) OFF

STRIG(n) STOP

n

may be 0, 2, 4, or 6, and indicates the button to be trapped as follows:

- 0 button A1
- 2 button B1
- 4 button A2
- 6 button B2

Remarks

STRIG(n) ON must be executed to enable trapping by the ON STRIG(n) statement (see "ON STRIG(n) Statement" in this chapter). After STRIG(n) ON, every time the program starts a new statement, BASIC checks to see if the specified button has been pressed.

If STRIG(n) OFF is executed, no testing or trapping takes place. Even if the button is pressed, the event is not remembered.

If a STRIG(n) STOP statement is executed, no trapping takes place. However, if the button is pressed it is remembered so that an immediate trap takes place when STRIG(n) ON is executed.

Returns the string representation of the value of a specified numeric expression.

Syntax STR\$(numexp)

Remarks For positive numbers, the string generated by STR\$ has a leading blank for the sign field.

See the complementary VAL function in this chapter.

Example 10 A\$ = STR\$(70) 20 PRINT A\$ RUN 70

Ok

70 (the argument of STR\$) is a number, but the contents of A\$ is a two character string whose value is "70".

Example

5 REM ARITHMETIC FOR KIDS 10 INPUT "TYPE A NUMBER";N 20 ON LEN(STR\$(N)) GOSUB 30,100,200,300,400,500

The entered number N is converted to a string by the STR\$ function. The program then branches according to the number of digits in the number entered.

Example

10 A! = 1.3 20 A# = VAL(STR\$(A!)) 30 PRINT A# RUN 1.3 Ok

The conversion in line 20 causes the value in A! to be stored accurately in the double-precision variable A#.

Returns a string of specified length whose characters all have the same ASCII code or equal the first character of a given string.

Syntax STRING\$ (length, code)

STRING\$ (length, stringexp)

length specifies the length of the resulting string (0-255).

code specifies the ASCII code of the character used to form the

resulting string (0-255).

stringexp is a string expression whose first character is used to form

the resulting string.

Example 10 X\$ + STRING\$(10,45)

20 PRINT X\$ "MONTHLY REPORT" X\$

RUN

____MONTHLY REPORT____

Exchanges the values of two variables.

Syntax

SWAP variable1, variable2

variable1 and variable2 are two variables of the same type (integer, single-precision, double-precision, or string).

Remarks

The two variables must be of the same type or a "Type Mismatch" error occurs. The second variable must already be defined or an "Illegal function call" error occurs.

Example

10 AS = "ONE": BS = "ALL": CS = "FOR"
20 PRINT AS CS BS
30 SWAP AS, BS
40 PRINT AS CS BS
RUN
ONE FOR ALL
ALL FOR ONE

Ok

SYSTEM

Command

Closes all open data files and returns to

MS-DOS.

Syntax SYSTEM

Remarks When a SYSTEM command is executed, all

open files are closed, and control is returned to MS-DOS. Your GWBASIC program is lost. If you entered GWBASIC through a Batch file from MS-DOS, the SYSTEM command returns

control to the Batch file.

Function

Tabs the cursor or the print head to a specified position, in PRINT, LPRINT, or PRINT# statements.

Syntax

TAB(n)

n

is an integer expression in the range 1 to 255.

Remarks

If the current cursor or print position is already beyond the specified value "n" TAB goes to that position on the next line. Space 1 is the leftmost position, and the rightmost position is the defined width.

If the value of "n" exceeds the defined width, the modulo operation is applied. For example, PRINT TAB(243) on a 40-column screen is the same as PRINT TAB(3), because 243 MOD 40+3.

A semicolon is assumed to follow the TAB function; thus GWBASIC does not add a carriage return if the TAB function is at the end of the list of data items.

Example

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

20 READ A\$,B\$

30 PRINT A\$ TAB(25) B\$

40 DATA "G. T. JONES","\$25.00"
NAME AMOUNT

G. T. JONES \$25.00 Ok Returns the tangent of the argument.

Syntax TAN(numexp)

numexp is a numeric expression representing the angle in radians.

Remarks TAN(numexp) is calculated in single precision (unless "/D" is supplied in the GWBASIC

command line).

If TAN overflows, the "Overflow" error message is displayed, machine infinity with the appropriate sign is supplied as the result,

and execution continues.

Example 10 Y = Q*TAN(x)/2

Statement and Function

The TIME\$ statement sets the current time. The TIME\$ function retrieves the current time

Syntax

TIME\$ = stringexp stringvar = TIME\$

stringexp

is a string expression indicating the time to be set.

stringvar

is a string variable in which the current time (8 character string) is returned.

Remarks

As a statement (TIME\$=stringexp):

"stringexp" is a string expression indicating the time in the form:

- hh (sets the hour; minutes and seconds default to 00), or
- hh:mm (sets the hour and minutes; seconds default to 00), or
- hh:mm:ss (sets the hour, minutes and seconds)

A 24 hour clock is used; therefore 8:00 p.m. would be entered as 20:00:00.

A leading zero may be omitted from any of the above values, but you must include at least one digit for each field higher than the lowest field set. For example, 00:00:42 is the same as 0:0:42, but :42 is incorrect.

Note that the time may also have been set by MS-DOS prior to entering GWBASIC.

As a function (stringvar=TIME\$):

The TIME\$ function returns an eight character string in the form hh:mm:ss, where hh is the hour (00 through 23), mm is minutes (00 through 59), and ss is seconds (00 through 59).

Examples

TIME\$ = "8:0"
Ok
PRINT TIME\$
08:00:04
Ok

The following program displays the current date and time on the 25th line of the screen and assigns the number of seconds after the minute to the variable SEC.

10 KEY OFF:SCREEN 0,0,0:CLS 20 LOCATE 25,5 30 PRINT DATES,,TIME\$ 40 SEC = VAL(MID\$(TIME\$,7,2))

Possible Errors

- An "Illegal function call" error is issued, if any of the values are out of range. The previous time is retained.
- A "Type mismatch" error is issued, if "stringexp" is not a valid string.

TIMER

Function

Returns a single-precision number indicating the seconds that have elapsed since midnight or system reset.

Syntax

TIMER

Remarks

TIMER is a numeric read-only function. It calculates fractional seconds to the nearest degree possible. It may not be used as a user variable.

Example

10 TIME\$ = "23:59:59" 20 FOR K = 1 TO 10

30 PRINT "TIMES = ";TIMES, "TIME = ";TIMER

40 NEXT 40 NEXT

TIMER {ON|OFF|STOP}

Statements

TIMER ON enables TIMER event trapping. TIMER OFF disables TIMER event trapping. TIMER STOP suspends TIMER event trapping.

Syntax

TIMER {ON|OFF|STOP}

Remarks

The TIMER ON statement enables real time event trapping by an ON TIMER GOSUB statement. While trapping is enabled with the ON TIMER GOSUB statement, GWBASIC checks between every statement to see if the timer has reached the specified level. If it has, the ON TIMER GOSUB statement is executed

TIMER OFF disables the event trap. If an event takes place, it is not remembered if a subsequent TIMER ON is used.

TIMER STOP disables the event trap, but if an event occurs, it is remembered and an ON TIMER GOSUB statement will be executed as soon as trapping is enabled.

Also see ON TIMER GOSUB statement in this chapter.

TRON/TROFF

Commands

TRON (TRACE ON) causes the line number of each statement executed to be listed.

TROFF (TRACE OFF) stops the line number listing initiated by TRON.

Syntax

TRON
TROFF

Remarks

The TRON statement (executed in either immediate or program mode) is used as a debugging tool, since it enables a trace flag that displays each line number of the program as it is executed. The numbers appear enclosed in square brackets. The trace flag is disabled with the TROFF statement (or when a NEW command is executed).

Example

```
Ok
10 K = 10
20 FOR J=1 TO 2
30 L=K + 10
40 PRINT J;K;L
50 K = K + 10
60 NEXT
70 END
Ok
TRON
Ok
RUN
 [10][20][30][40] 1 10 20
 [50][60][30][40] 2 20 30
 [50][60][70]
Ok
```

Calls a machine language subroutine.

Syntax

USR [n](argument)

n

is in the range 0 to 9 and corresponds to the digit supplied with the DEF USR statement for that routine. If omitted USR 0 is assumed.

argument

is the value passed to the subroutine. It may be any numeric or string expression. Even if the subroutine does not require an argument, a dummy argument must be supplied.

Remarks

The type (numeric or string) of the variable receiving the function call must be consistent with that of the argument passed (see Chapter 6).

Prior to calling each USR function, a corresponding DEF USR statement must be executed to define the USR function call offset. This offset and the currently active DEF SEG address determine the starting address of the subroutine.

The CALL statement is another way to call a machine language subroutine.

USR Function

Example

100 DEF SEG = &H8000 110 DEF USRO = 0 120 X = 5 130 Y = USRO(X) 140 PRINT Y

Calls a machine language subroutine at 8000H. It passes 5 as an argument and returns a value in Y.

Converts the string representation of a number to its numeric value.

Syntax

VAL (stringexp)

Remarks

VAL function strips leading blanks, tabs, and linefeeds from the argument string.

The remaining string is converted to a number, if it is a valid numeric representation, otherwise VAL returns 0 (zero). For example:

VAL ("-3")

returns -3.

VAL ["ABC"]

returns 0.

See the STR\$ function in this chapter for numeric-to-string conversion.

Example

OK
PRINT VAL("394 LOWELL ST")
394
OK

Returns the memory address of the variable or

file control block.

Syntax 1 VARPTR (variable)

Syntax 2 VARPTR (#filenum)

variable is the name of a numeric or string variable in the program.

filenum is the number under which the file was opened.

Remarks For both formats, the address returned is an

integer in the range 0 to 65535. This number is the offset into GWBASIC's Data Segment. The address is not affected by the DEF SEG

statement.

Syntax 1

Returns the address of the first byte of data identified with "variable".

A value must be assigned to "variable" prior to execution of VARPTR. Otherwise an "Illegal function call" error results. Any type of variable may be used (numeric, string).

VARPTR is usually used to obtain the address of a variable or array so that it may be passed to a machine language subroutine. A function call of the form VARPTR(A(0)) is usually specified when passing an array, so that the lowest-addressed element of the array is returned.

All simple variables should be assigned before calling VARPTR for an array, because the addresses of the arrays change whenever a new simple variable is assigned.

VARPTR Function

Syntax 2

Returns the starting address of the file control block for the specified file.

Example

10 X = USR(VARPTR(Y))

110 OPEN "A:FILEA.DAT" AS #2 120 GET #2 'get address of FCB 130 FCBADR = VARPTR(#2) Returns a character form of the memory address of the variable.

Syntax VARPTR\$ (variable)

variable is a variable existing in the program.

Remarks A value must be assigned to "variable" prior

to execution of VARPTR\$. Otherwise, an "Illegal function call" error results. Any type of variable (numeric, string) may be used.

VARPTR\$ returns a three-byte string in the form:

byte 0 = type

byte 1 = low byte of address

byte 2 = high byte of address

Note that type indicates the variable type:

2 integer

3 string

4 single-precision

8 double-precision

Because array addresses change whenever a new simple variable is assigned, always assign all simple variables before calling VARPTR\$ for an array element.

The returned value is the same as:

CHR\$(type)+MKI\$(VARPTR(variable))

You can use VARPTR\$ to indicate a variable name in the command string for DRAW. For example:

DRAW "0=1;"

or

DRAW "0 = " + VARPTR\$(I)

Defines subsets of the screen called "viewports;" into these, window contents will be mapped. (Graphics Mode only).

Syntax

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

(x1,y1)-(x2,y2)

represent the 'x' and 'y' coordinates within the physical boundary of the screen that graphics will map into. (x1,y1) are the upper-left, and (x2,y2) the lower-right coordinates of the viewport defined.

color

permits the defined viewport to be filled with a specified color. If 'color' is omitted then the viewport is not filled-in.

border

permits the drawing of a border-line around the viewport (if the necessary space is available). If border is omitted, no border-line is drawn.

VIEW Statement

Remarks

Initially, RUN or VIEW with no arguments define the entire screen as the viewport.

For the form:

VIEW (x1,y1) - (x2,y2)

all points plotted are relative to the viewport. That is, "x1" and "y1" are added to the x and y coordinates before putting down the point on the screen.

If:

VIEW (10,10) - (200,100)

were executed, then the point set down by the statement PSET(0,0),3 would actually be at the physical screen location 10,10.

For the form:

VIEW SCREEN (x1,y1)-(x2,y2)

all coordinates are absolute and may be inside or outside of the screen limits, but only those within the VIEW limits will be plotted.

If:

VIEW SCREEN (10,10)-(200,100)

were executed, then the point set down by the statement PSET(0,0),3 would actually not appear because 0,0 is outside of the viewport. PSET(10,10),3 is within the viewport, and places the point in the upper-left hand corner of the viewport.

VIEW with no arguments defines the entire viewing surface as the viewport. This is equivalent to VIEW (0,0)-(319,199) in medium resolution, VIEW (0,0)-(639,199) in high resolution, and VIEW (0,0)-(639,399) in super resolution.

Multiple viewports can be defined, but only one viewport (called the "current viewport") may be active at any one time. Each time a VIEW statement is executed a viewport is defined and this is the current viewport. Thus, to change the current viewport, you have to execute another VIEW statement.

A number of VIEW statements may be executed. If the newly described viewport is not wholly within the previous viewport, the screen can be re-initialized with the VIEW statement with n arguments. Then the new viewport may be stated. If the new viewport is entirely within the previous one, as the first of the following examples, the intermediate VIEW statement is not necessary.

RUN and SCREEN will disable the viewports.

VIEW and WINDOW statements allow you to do scaling by changing the size of your viewport. A large viewport will make your objects large and a small viewport will make your objects small. (Refer to "WINDOW Statement" in this chapter.)

Example 1

This example opens three viewports, each smaller than the previous one. In each case, a line that is defined to go beyond the borders is programmed, but appears only within the viewport border.

```
260
       CLS
       VIEW: REM ** Make the viewport the entire
280
       VIEW (10,10) - (300,180),,1
320
330
       LINE (0,0) - (310,190),1
360
       LOCATE 1,11: PRINT "A big viewport"
380
       VIEW SCREEN (50,50)-(250,150),,1
       CLS:REM** Note, CLS clears only viewport
400
420
       LINE (300,0)-(0,199),1
440
       LOCATE 9,9: PRINT "A medium viewport"
       VIEW SCREEN (80,80)-(200,125),,1
460
480
       CLS
500
       CIRCLE (150,100),20,1
       LOCATE 11.9: PRINT "A small viewport"
520
```

This example demonstrates scaling with VIEW and WINDOW.

Example 2 10 KEY OFF:CLS:SCREEN 1,0:COLOR 0, 0 20 WINDOW SCREEN(0,0)-(320,200) 30 GOSUB 70:FOR K = 1 TO 1000:NEXT :CLS 40 VIEW (1,1)-(160,90),,2:GOSUB 70 50 'Make it small 60 GOTO 100 70 'Create the picture 80 CIRCLE (160,100),60,1,,,1 90 RETURN 100 END

The following example defines two viewports:

Example 3	10	SCREEN 1:VIEW:CLS:KEY OFF
	20	VIEW (1,1)-(151,91),,1
	30	VIEW (165,1)-(315,91),,2
	40	LOCATE 2,4:PRINT "Viewport 1"
	50	LOCATE 2,25:PRINT "Viewport 2"
	60	VIEW (1,1)-(151,91):GOSUB 500
	70	VIEW (165,1)-(315,91):GOSUB 1000
	80	END
	500	'Draw a circle in first viewport
	510	CIRCLE (65,50),30,2
	520	RETURN
	1000	'Draw a line in second viewport
	1010	LINE (45,50)-(90-75),1,8
	1020	DETLIDA

Sets the boundary of the text window.

Syntax VIEW PRINT [line1 TO line2]

line1 is the top line of the text window

line2 is the bottom line of the text window

Remarks Statements and functions which operate

within the text window include CLS, LOCATE, and the SCREEN function. The Screen Editor will limit functions such as scroll and cursor movement to the text window.

If no parameters are specified, VIEW PRINT will initialize the text window to include the whole screen

Example VIEW PRINT 1 TO 5

creates a text window of 5 lines on the top of the screen.

Suspends program execution while monitoring the status of a machine input port. WAIT may only be used in a program.

Syntax

WAIT port, i [, j]

port

is the port number, in the range 0 to 65535

i, j

are integer expressions in the range 0 to 255

Remarks

The WAIT statement causes execution to be suspended until a specified machine input port develops a specified bit pattern. The data read at the port is XORed with the integer expression "j" and then ANDed with "i". If the result is zero, GWBASIC loops back and reads the data at the port again. If the result is nonzero, execution continues with the text statement. If "j" is omitted, it is assumed to be zero.

Note

It is possible to enter an infinite loop with the WAIT statement.

You can do a CTRL-BREAK or a System Reset to exit the loop.

Example

100 WAIT 32, 2

Loop through a series of statements as long as a given condition remains true.

Syntax

WHILE condition loop statements WEND

condition

is a numeric, relational or logical expression. GWBASIC determines whether the condition is true or false by testing the result of the expression for non zero and zero, respectively. A non zero result is true and a zero result is false. Because of this, you can test whether the value of a variable is non zero or zero by merely specifying the name of the variable as a condition.

loop

statements are executed until a WEND statement is encountered $\,$

Remarks

If "condition" is not zero (i.e., true), "loop statements" are executed until the WEND statement is encountered. GWBASIC then returns to the WHILE statement and checks "condition". If it is still true, the process is repeated. If it is zero (i.e. false), execution resumes with the statement following the WEND statement. WHILE/WEND loops may be nested to any level. Each WEND will match the most recent WHILE. An unmatched WHILE statement causes a "WHILE without WEND" error, and an unmatched WEND statement causes a "WEND without WHILE" error.

Do not direct program flow into a WHILE/WEND loop without entering through the WHILE statement.

WHILE . . . WEND Statements

Example	90N	'BUBBLE SORT ARRAY A\$
•	100	FLIPS = 1 'FORCE ONE PASS
	110	WHILE FLIPS
	115	FLIPS = 0
	120	FOR i = 1 TO J-1
	130	IF A\$(I)>A\$(I+1) THEN 150
	133	FLIPS = 1
	135	SWAP A\$(I), A\$(I+1)
	140	NEXT I
	150	WEND

Sets the line width in characters. GWBASIC adds a carriage return after outputting the specified number of characters.

Syntax 1 WIDTH [LPRINT] size

Syntax 2 WIDTH filenum, size

Syntax 3 WIDTH device, size

size is an integer expression in the range 0 to 255. It specifies the

new width. WIDTH 0 is the same as WIDTH 1.

filenum is a numeric expression in the range 1 to 15. This is the

number of a file OPENed to one of the devices listed below.

device is a string expression returning the device identifier. Valid devices are: SCRN:, LPT1:, LPT2:, LPT3:, COM1:, COM2:,

COM3:, or COM4:.

WIDTH LPRINT size

Sets the line width at the line printer.

WIDTH size or WIDTH "SCRN:".size

Sets the screen width (in Text mode), selects a text window or changes mode (in Graphics mode). Changing the screen or text window width, or the mode, causes the screen to be cleared.

In Text Mode (mode 0) "size" may only have the values 40 or 80, selecting either a 40column or an 80-column screen.

In Graphics Mode (mode 1, 2, or 100) you can either change mode or select a text window of width 40 or 80. The width of the function key display will correspond to the selected width. If the number of columns displayed is 40 you may enter CTRL-T to scroll the function key display horizontally.

The following summarizes all possible cases.

0 (text)	40	select a 40-column screen
(UCAU)	80	select an 80-column screen
	80	place the system in high- resolution (mode 2)
1 (medium-res)	40	create a test window of width 40
	80	forces the screen into high resolution
2 (high-res)	40	create a text window of width 40
	80	create a text window of width 80
100 (super-res)	40	create a text window of width 40
	80	create a text window of width 80

WIDTH filenum, size

Changes the width of the device associated with "filenum" to the new "size" specified. This form of the WIDTH statement has meaning only for: LPT1:, LPT2:, LPT3:, COM1:, COM2:, COM3:, and COM4:. This allows the width to be changed while the file is open.

WIDTH device, size

Stores the new 'size' without changing the current width, if the device is already open. A subsequent OPEN device FOR OUTPUT AS # n will use this value of "size" for width as long as the file is open.

Note that LPRINT, LLIST and LIST, "LPTn" do an implicit open and are therefore affected by this statement.

Remarks

When the WIDTH statement causes a change in the screen mode, colors are set to their default values.

You should turn the function key display off when changing the window width (by a KEY OFF statement), otherwise, if the width is decreased, part of the old (wider) function key display may be left on the screen.

If "size" is 255, the line width is "infinite"; that is, GWBASIC never inserts a carriage return. However, the position of the cursor or the print head, as given by the POS or LPOS function, returns to zero after position 255. WIDTH 255 is the default for communications files.

Changing the width for a communications file does not alter the receive buffer, it just tells GWBASIC to send a carriage return after every "size" character.

Possible Errors

If "size" is outside the above specified ranges, an "Illegal function call" error is returned. The previous value is retained.

WIDTH Statement

Example

- 10 WIDTH "LPT1:", 5
- 20 OPEN "LPT1:" FOR OUTPUT AS 1
- 30 PRINT #1, "1234567890"
- 35 LPRINT
- 40 WIDTH #1, 6
- 50 PRINT #1, "1234567890"

will yield on the printer

12345

67890

123456 7890 Permits the redefinition of the screen coordinates. (Graphics Mode only.)

Syntax

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

(x1,y1)-(x2,y2) (x1,y1) represent the upper-left coordinates of the window. (x2,y2) represent the lower-right coordinates of the window. These coordinates may be any single precision floating point number.

Remarks

WINDOW allows you to draw lines, graphs, or objects in space not bounded by the physical limits of the screen. This is done by using arbitrary programmer-defined coordinates called "world coordinates."

A world coordinate is any valid single precision floating point number pair. GWBASIC then converts world coordinate pairs into the appropriate physical coordinate pairs for subsequent display within screen space. To make this transformation from world space to the physical space of the viewing surface (screen), GWBASIC must know what portion of the unbounded (floating point) world coordinate space contains the information you want to be displayed.

This rectangular region in world coordinate space is called a window.

WINDOW defines the "window" transformation from x1,y1 (upper left x,y coordinates) to x2,y2 (lower right x,y coordinates). The x and y coordinates may be any single precision floating point number and define the "World Coordinate Space" that graphics will map into the physical coordinate space, as defined by the VIEW statement.

Initially, RUN, or WINDOW with no arguments, disables "Window" transformation.

WINDOW inverts the "y" coordinate on the subsequent graphics statement. This allows the screen to be viewed in true cartesian coordinates. The WINDOW SCREEN variant does not invert the "y" coordinate.

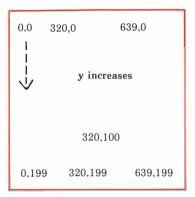
In the physical coordinate system, if you run the following:

NEW

or

SCREEN 2

the screen will appear with standard coordinates as:

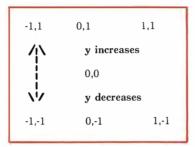


If a window command is issued with SCREEN omitted, the screen is viewed in the CARTESIAN coordinates.

For example if:

WINDOW (-1,-1)-(1,1)

was executed then the screen appears as:



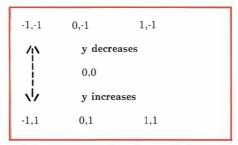
Note now that the "y" coordinate is inverted so that (x1,y1) is the lower-left coordinate and (x2,y2) is the upper-right coordinate.

If the SCREEN attribute is included then, the coordinates are not inverted. So that, (x1,y1) is the upper-left coordinate and (x2,y2) is the lower-right coordinate.

For example:

WINDOW SCREEN (-1,-1)-(1,1)

appears as:



All possible pairings of "x" and "y" are valid. A restriction is that "x1" cannot equal "x2" and "y1" cannot equal "y2".

The WINDOW statement uses a process called "clipping", whereby pixels which are referenced outside a coordinate range are excluded from the viewing area. Any object lying partially within and partially without a coordinate range is clipped so that only the pixels referenced in range will appear.

WINDOW also features a "zoom in"/"zoom out" facility. Choosing window coordinates larger than an image will display the entire image, but the image will be small. Choosing window coordinates smaller than an image will cause clipping, allowing only a portion of the image to be displayed and magnified. By specifying small and large window sizes, you can zoom in until an object occupies the entire screen, or you can zoom out until the image is nothing but a spot on the screen.

RUN, SCREEN, and WINDOW with no attributes will disable any WINDOW coordinates and return the screen to physical coordinates.

Examples

The following example demonstrates image clipping.

- 10 SCREEN 100
- **20 CLS**
- 30 WINDOW (-6,-6)-(6,6)
- 40 CIRCLE (4,4),5,1
- 50 'the circle is large only part is visible
- 60 WINDOW (-100,-100)-(100,100)
- 70 CIRCLE (4,4),5,1 'the circle is small
- 80 END

The following example shows the effect of zooming.

KEY OFF:CLS:SCREEN 1.0 10 X = 1000:WINDOW (-X,-X)-(X,X):R = 2030 'create a graph with large coord range 40 GOSUB 1000:FOR K = 1 TO 1000:NEXT:CLS X = 60:WINDOW (-X,-X)-(X,X):R = 2060 'smaller coord range increases circle size 70 GOSUB 1000:FOR K = 1 TO 1000:NEXT:CLS 80 X = 100:WINDOW (-5,-5)-(X,X): R = 20'modify window to show only portion of axes 90 100 GOSUB 1000:FOR K = 1 TO 1000:NEXT :CLS 110 PRINT " ZOOMING " 120 CLS:T = -50:U = 100:X = U 130 FOR K = 7 TO 1500:NEXT 140 FOR K = 1 TO 45 150 T=T+1:U=U-1:X=X-1:R=20 160 WINDOW (T,T)-(U,U):CLS:GOSUB 1000 170 **NEXT K** 180 END 1000 'Subroutine display 1010 LINE [X,0]-[-X,0],,,&HAA00 'create x axis 1020 LINE (0,X)-(0,-X),,,&HAA00 'create y axis 1030 CIRCLE (X/2,X/2),R 'circle has radius r 1040 FOR K = 1 TO 50:NEXT 'delay' 1050 RETURN

The following example illustrates two lines with the same endpoint coordinates. The first is drawn on the default screen, and the second is on a redefined window.

```
200 LINE (100,100) - (150,150), 1
220 LOCATE 2,20:PRINT "The line on the default screen"
240 WINDOW SCREEN (100,100)-(200,200)
260 LINE (100,100) - (150,150), 1
280 LOCATE 8,18:
300 PRINT "& the same line, new window"
```

Writes data to the screen.

Syntax

WRITE [list_of_expressions]

list-of-expressions

list-of-numeric and/or string expressions. They must be separated by commas.

Remarks

If "list of expressions" is omitted, a blank line is output. If "list-of-expressions" is included, the values of the expressions are output on the screen.

When the values of the expressions are output, each item is separated from the last by a comma. Strings are delimited by quotation marks. After the last item in the list is displayed, GWBASIC inserts a CR LF.

WRITE and PRINT are similar. The difference between WRITE and PRINT is that WRITE inserts commas between the items on the screen and delimits strings with quotation marks. Also numbers are not preceded by blanks.

Example

10 A = 80:B = 90:C\$ = "THAT'S ALL" 20 WRITE A,B,C\$ RUN 80, 90,"THAT'S ALL" Ok



Writes data to a sequential file.

Syntax WRITE#filenum,list-of-expressions

filenum is the number under which the file was OPENed in "O"

mode (see "OPEN" Statement in this chapter).

list-of-expressions

list of string or numeric expressions. They must be sepa-

rated by commas.

Remarks The difference between WRITE# and PRINT#

is that WRITE# inserts commas between the items as they are written to the file and delimits strings with quotation marks. Also, WRITE# does not precede positive numbers with blanks. Therefore, it is not necessary for the user to put explicit delimiters in the list. A

CR LF sequence is inserted after the last item

in the line is written to the file.

Example

10 A\$ = "CAMERA" : B\$ = '93604-1' 20 WRITE#1,A\$,B\$

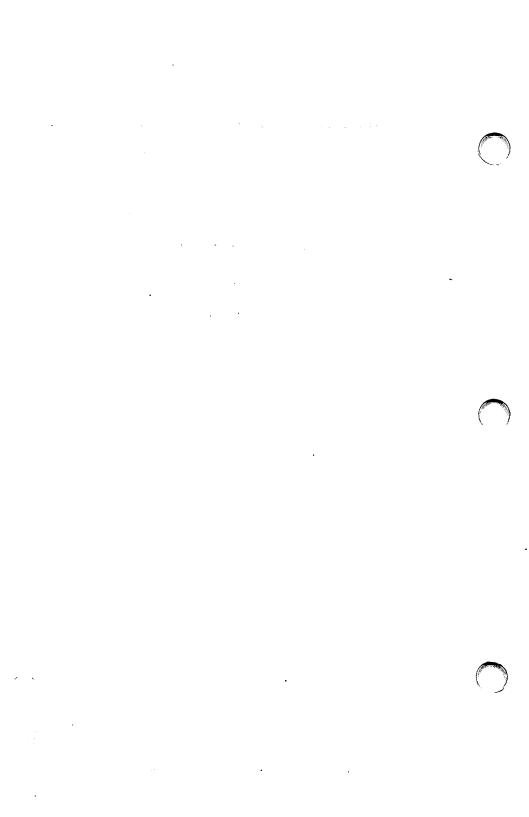
Statement 20 writes the following image to disk:

"CAMERA","93604-1"

A subsequent INPUT# statement, such as

30 INPUT#1,A\$,B\$

would input "CAMERA" to A\$ and "93604-1" to B\$.



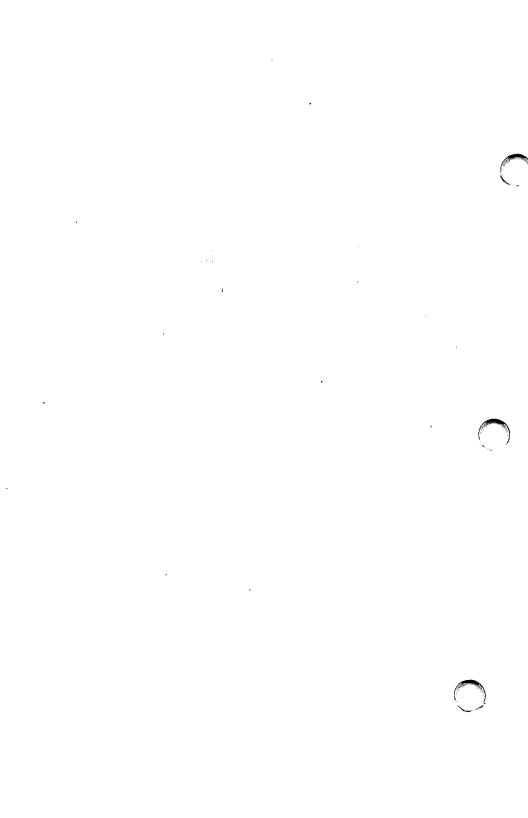
Contents

1- 1

DEB Capabilities

Introduction

	16-Color Graphics Look-Up Table (LUT) Overlay Modes	1- 3 1- 5 1- 6
2	How to Program the DEB Programming Steps	2- 1
3	DEB Statements Overview SCREEN Statement COLOR Statement PALETTE and PALETTE USING Statements Default Palettes Blinking Color Effects for DEB Palettes 0-3 Dither Combinations for DEB Palettes 0-3 Remarks Examples	3- 1 3- 2 3- 4 3- 7 3-10 3-13 3-14 3-15 3-16
4	Programming the LUT Overview 16-Color Graphics LUT Programming Overlay Modes LUT Programming	4- 1 4- 2 4-22



1 DEB Capabilities

- Introduction
- 16-Color Graphics
- Look-Up Table (LUT)
- Overlay Modes

.

INTRODUCTION

The Display Enhancement Board option (DEB) adds improved color and graphics functionality to your AT&T PC 6300. When you use the DEB with the PC 6300 color monitor, you can display graphics in up to 16 colors simultaneously or display text-on-graphics or graphics-on-graphics overlays. When you use the DEB with the PC 6300 monochrome monitor, you have the same capabilities as you do with the color monitor, except that colors are displayed as "shades of green."

The DEB is compatible with existing software, so that all the programs you have already can be used now as if the DEB were not installed. Of course, these programs do not have access to any of the new capabilities.

The purpose of this supplement to the GWBASIC Programmer's Guide is to give you the information you need to take complete advantage of the DEB's capabilities. It assumes that you are familiar with video programming in GWBASIC. If you are not, read the chapter on Graphics, and the portions of the Command Reference that discuss graphics statements, in the GWBASIC Programmer's Guide.

Before you begin writing programs for the DEB, follow the procedures in the DEB Installation Manual for installing the DEB hardware and device driver software.

The DEB is an optional hardware component for the AT&T PC 6300 that works in conjunction with the PC 6300's built-in Video Display Controller (VDC) to provide improved color and graphics functionality.

The built-in VDC contains circuitry and memory that supports either 4 color medium resolution $(320 \times 200 \text{ pixels})$ graphics, 1 color high resolution $(640 \times 200 \text{ pixels})$ graphics, or 1 color super resolution $(640 \times 400 \text{ pixels})$ graphics.

The DEB contains additional circuitry and memory that can be combined with the capabilities of the built-in VDC to produce up to 16 colors in either high or super resolution. You can also program the VDC and DEB separately, treating them as two separate images which are combined on one screen to produce text-on-graphics or graphics-ongraphics overlays. These overlay modes let you use up to 8 colors.

16-COLOR GRAPHICS

This feature lets you display 16 colors in either high resolution (640×200) or super resolution (640×400). Not only can you use the standard 16 colors, you can also combine colors to form new colors and cause pixels to blink from one color to another.

The DEB provides 5 palettes for you to use when programming in color. At any point in your program, you select one of the palettes as the "active" palette. The color combinations contained in that palette determine what colors and effects show on the screen.

Each of the first 4 palettes contains a default set of 16 color combinations, but to suit the needs of your program you can change the contents of the palette to any one of the following:

 any of the 16 standard colors with which you are already familiar from the standard applications.
 The standard colors are:

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

- a mixture, or "dithering," of any 2 of the 16 standard colors
- an alternation, or blinking, between any 2 of the standard 16 colors

The fifth palette contains no default combinations. You program the fifth palette by loading color values into a 256-element array of integers. GWBASIC uses this special palette to program the DEB's color look-up table (LUT).

LOOK-UP TABLE (LUT)

The LUT resides in RAM on the DEB board. The LUT contains 256 values that determine the colors, blinking, and dithering that appear on the screen. Whether you need to learn about the use and layout of the LUT depends on the application you are writing.

If you use the standard palettes, you need not be concerned with the LUT. GWBASIC automatically programs the LUT to correspond to the way you set up the palettes.

If you program a custom LUT, you greatly increase the color combinations and blinking effects available to you.

OVERLAY MODES

These modes let you display text-on-graphics or graphics-on-graphics images by treating the VDC and DEB as separate entities that write to the same screen. In the overlay modes, the output of the VDC takes precedence over the output of the DEB. If you program the VDC and DEB to display different attributes at the same pixel, the attributes selected by the VDC are displayed.



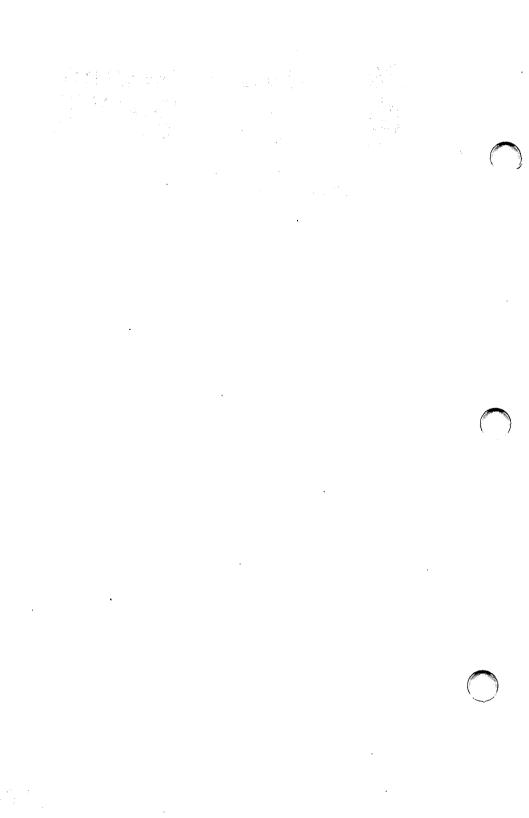
You can use either of two text-on-graphics modes. In one, you can program the DEB to display high resolution graphics in up to 8 colors; in the other, you can program the DEB to display super resolution graphics in up to 8 colors. In both, the VDC displays 25 lines of 80 characters each.

You can select either of two graphics-on-graphics modes. One mode uses the VDC to display high resolution graphics in one color while the DEB displays high resolution graphics in up to 8 colors. The other mode uses the VDC for super high resolution graphics in one color and the DEB for super high resolution graphics in 8 colors.

The overlay modes offer 5 palettes. Each of the first 4 palettes has 8 positions. These four palettes have default colors that you can change to suit your needs. You can choose 8 color combinations from any of the 16 standard colors, or blink between 2 of the standard colors. The dithering combinations of the 16-color graphics modes are not available. You can also use the fifth palette to custom program the LUT.

How to Program the DEB

Programming Steps



PROGRAMMING STEPS

There are three steps for video programming in GWBASIC, which apply whether or not you are using the DEB capability:

- 1 Set the video mode by using the SCREEN statement.
- **2** Select the color combinations and effects you want to use.
- **3** Construct the graphics images you want to display.

This chapter describes each of these steps in detail. This chapter does **not** describe how to use the fifth palette to program the LUT directly. (See **Chapter 4, Programming the LUT.**)

Setting Mode and Page

As in standard GWBASIC, you use the SCREEN statement to select an operating mode. If you are using one of the overlay modes, the SCREEN statement also selects the active page, which determines whether the VDC or the DEB receives the output of PRINT or graphics display statements. The VDC is page 0 and the DEB is page 128. In the text-on-graphics modes, all text output statements default to page 0 and all graphics display statements default to page 128. If you want text to appear on the DEB graphics screen, you must issue a SCREEN statement that sets the active page to 128 before you display the text.

Setting Colors and Effects

Colors and effects are controlled by two statements: COLOR and PALETTE. The COLOR statement syntax extends the standard GWBASIC COLOR statement, allowing you to select background and foreground default colors and to select the active palette. The PALETTE statement is new. You use PALETTE to program color combinations into the active palette or to reset the active palette to its default assignments. A form of the statement, PALETTE USING, allows you to reprogram the entire active palette at once by specifying an integer array that contains the new values. Tables of the available color combinations and the default values for each palette are in the next chapter on DEB Statements.

Displaying Graphics Images

You use the same statements for DEB graphics as you do for normal GWBASIC graphics. However, in normal GWBASIC statements, you specify the color number to be used in drawing a line or circle. For DEB graphics, you specify the *pelette position* in the active palette that contains the color combination or effect you want to use. For example, you could select 16-color super resolution mode, select palette 1 as the active palette, and draw a red circle, with the following code fragment:

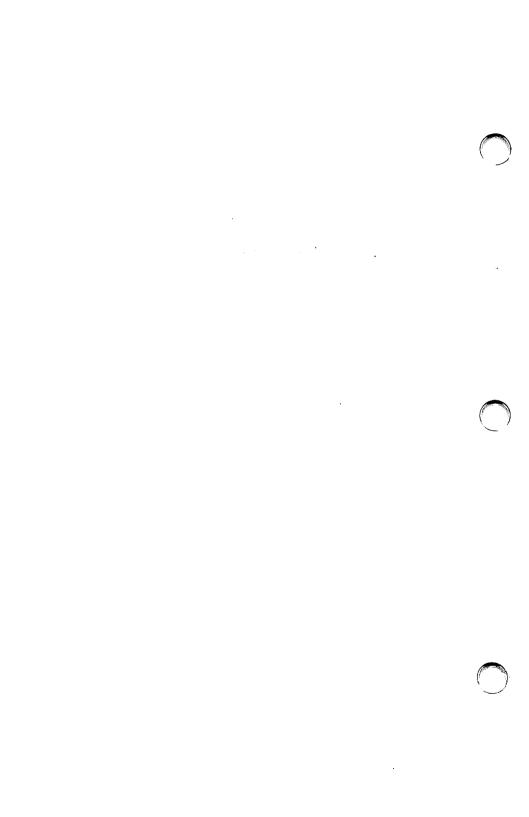
10 SCREEN 102 20 REM 30 COLOR ,,1 40 REM 50 CIRCLE (320,200),100,2 60 REM 'select 640 × 400
'16-color mode
'set active
'palette to 1
'the default color in
'position 2 is red

• •

3

DEB Statements

- Overview
- SCREEN Statement
- COLOR Statement
- PALETTE and PALETTE USING Statements
- Default Palettes
- Blinking Color Effects for DEB Palettes 0-3
- Dither Combinations for DEB Palettes 0-3
- Remarks
- Examples



OVERVIEW

This chapter gives detailed descriptions of the GWBASIC statements that you can use for DEB graphics programming.

If you plan to use Palette 4, the LUT palette, carefully read ${\bf Chapter~4}$ before you begin using the statements in this chapter to program the LUT.

SCREEN STATEMENT

SCREEN

The SCREEN statement establishes the mode for the display and lets you select the active display page. SCREEN also selects and initializes Palette 0 as the active palette when you enter a new mode.

Syntax

SCREEN [mode][,dummy1][,apage][,dummy2]

mode

is an integer expression which evaluates to one of the following:

- 101 16-color graphics with a resolution of 640×200 .
- 102 16-color graphics with a resolution of 640×400 .
- 103 an overlay mode. The DEB image is 8-color graphics with 640×200 resolution. The VDC image is 80 character by 25 line text.
- an overlay mode. The DEB image is 8-color graphics with 640×400 resolution. The VDC image is 80 character by 25 line text.
- 105 an overlay mode. The DEB image is 8- color graphics with 640×200 resolution. The VDC image is 1-color graphics with 640×200 resolution.
- 106 an overlay mode. The DEB image is 8-color graphics with 640×400 resolution. The VDC image is 1-color graphics with 640×400 resolution.

dummy1

is ignored, but is allowed for compatibility with non-DEB syntax.

apage

selects the active page, i.e., the page to be written to by output statements to the screen. Apage is an integer expression that results in a value of 0 or 128. Page 0 is the VDC page and page 128 is the DEB page.

In the two 16-color graphics modes (101 and 102), the active page is always zero.

dummy2

is ignored, but is allowed for compatibility.

Examples

SCREEN 105,,128

'Selects a graphics-on-graphics overlay mode, with all subsequent output sent to the DEB page.

SCREEN ,,0

'Do not change modes, but send subsequent output to the VDC page.

COLOR STATEMENT

COLOR

The COLOR statement sets the background and foreground colors and selects the active palette. The syntax for the COLOR statement varies according to the mode you select with the SCREEN statement.

Syntax 1 (Modes 101,102)

COLOR [DEBfg][,DEBbg][,palette]

Syntax 2 (Modes 103,104)

COLOR [DEBfg][,DEBbg][,VDCfg]
[,VDC bg][,palette]

Syntax 3 (Modes 105,106)

COLOR [DEBfg][,DEBbg][,VDCfg][,palette]

DEBfg (foreground)

is an integer expression in the range 1-7 for overlay modes and 1-15 for 16-color graphics modes. DEBfg identifies the position in the active palette which controls the color combination or effect of subsequent output to the screen. The color combination or effect in the DEBfg position will be used for writing text to the screen, and also for the output of graphics statements unless some other position is specified in the graphics statement itself.

When you enter a DEB mode, DEBfg is set to a default of 7. If you do not enter a value for DEBfg, it does not change from the value set by the last COLOR statement.

DEBbg (background)

is an integer expression in the range 0-255 which defines the color combination or effect to be used for palette position 0. This is the background, or color displayed when the value of the DEB image for a particular pixel is 0. (See tables of combinations in next section on PALETTE statement.) When you enter a DEB mode, DEBbg defaults to 0 (black).

VDCfg

is an integer expression in the range of 0-15 for graphics and 0-31 for text that specifies the color for the VDC foreground. When you enter an overlay mode, VDCfg defaults to 7 (white).

VDCbg

is an integer expression in the range 0-15 that specifies the VDC background when displaying characters in text mode. VDCbg defaults to 0 (black) when you enter an overlay mode.

palette

is an integer expression that sets the active palette. Valid ranges are 0-3 for the standard palettes and 4 for the LUT palette. If you omit palette from the COLOR statement, the active palette does not change.

Remarks

The values you specify in DEB COLOR statements fall into three categories:

• a color selection for the VDC from the same ranges as you use in the standard text mode. These selections produce the same effect on the screen as they do in the standard (non-DEB) text mode.

- a color selection for the DEB foreground. Here you specify a palette position instead of a color number. GWBASIC then looks up the color combination or effect in the palette position you've specified, and uses it in the PRINT statements and some of the graphics statements that follow the COLOR statement. If the syntax of a particular graphics statement includes a parameter for specifying a palette position, that value overrides the position specified in the COLOR statement.
- specification of the DEB background based on a color combination from the tables following the PALETTE statement in this chapter. You can also set the DEB background by using the PALETTE statement to change Palette position 0.

PALETTE AND PALETTE USING STATEMENTS

PALETTE Use this statement to set values in palettes and

reset palettes to their default values.

Syntax 1 PALETTE

Syntax 2 PALETTE [position][,value]

Syntax 3 PALETTE USING array (array index)

Remarks The PALETTE and PALETTE USING statements

work on the active graphics page and on the active

palette.

Syntax 1 sets the active palette to its default val-

ues. (See the following tables.)

Syntax 2 lets you change the values in the active

palette, one palette position at a time.

position is an integer expression which identifies the posi-

tion to be changed. If the active palette is 0-3, then the valid range for position is 0-15 for 16-color graphics modes and 0-7 for overlay modes. For Pal-

ette 4, the valid range for position is 0-255.

value is an integer expression which identifies the color

combination or effect to be programmed into the selected position in the active palette. For Palettes 0-3, valid values range from 0-255. For Palette 4, valid values range from 0-15 and values greater

than 15 are treated modulo 16.

Syntax 3 lets you set all the values in the active

palette with one statement.

array

is an integer array of at least 256 elements.

array index is an integer expression which defines the element within the specified array at which palette programming begins. At least 256 elements must follow this element.

Standard Palettes (0-3)

The first 8 or 16 elements of the array are loaded into the active palette. The entire active palette is reprogrammed based on the values in the array. The array values range from -1 to 255. Values greater than 255 are treated modulo 256. A value of -1 specifies that the value in the corresponding palette position not be changed. The values from 0 to 255 come from the tables at the end of the chapter.

NOTE: Dimension the array to have 256 elements even though only 8 or 16 are used for the standard palettes.

The LUT Palette (Palette 4)

All 256 elements are used to program the LUT directly. Valid values are in the range -1 to 15. Values greater than 15 are treated modulo 16. A value of -1 specifies that the value in the corresponding position in the LUT not be changed, and values 0-15 represent the standard 16 colors.

In Syntax 2 and Syntax 3, if you specify a palette position greater than the value allowed for the mode in which you are working, the value you specify will be put in that palette's highest position. For example, if you attempted to set palette position 13 to red when working in overlay mode, which has 8-position palettes, the **8th** palette position would be set to red.

DEFAULT PALETTES

The defaults for each of the four palettes are:

Palette Number 0

Position	Color
0	0 = black
=	0 10.000
1	2 = green
2	4 = red
3	6 = brown
4	1 = blue
5	3 = cyan
6	5 = magenta
7	7 = white
8	8 = gray
9	9 = light blue
10	10 = light green
11	11 = light cyan
12	12 = light red
13	13 = light magenta
14	14 = yellow
15	15 = high-intensity white

Palette Number 1					
Position	Color				
0	0 = black				
1	3 = cyan				
2	5 = magenta				
3	7 = white				
4	1 = blue				
5	2 = green				
6	4 = red				
7	6 = brown				
8	8 = gray				
9	9 = light blue				
10	10 = light green				
11	11 = light cyan				
12	12 = light red				
13	13 = light magenta				
14	14 = yellow				
15	15 = high-intensity white				

Palettes 2 and 3 are the same, and they contain the standard colors in numerical order.

Palette Number 2 and Palette Number 3

	amber = ama r arette mar
Position	Color
0	0 = black
1	1 = blue
2	2 = green
3	3 = cyan
4	4 = red
5	5 = magenta
6	6 = brown
7	7 = white
8	8 = gray
9	9 = light blue
10	10 = light green
11	11 = light cyan
12	12 = light red
13	13 = light magenta
14	14 = yel low
15	15 = high-intensity white

BLINKING COLOR EFFECTS FOR DEB PALETTES 0-3

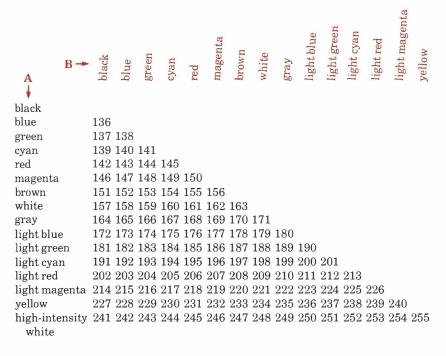
Color combinations 16-135 have been pre-assigned to allow you easy access to blinking effects while using the standard palettes. The following table describes the available combinations.

															wh
	B → B	green	cyan	red	magenta	brown	white	gray	light blue	light preen	light evan	light red	light magenta	yellow	high-intensity
A ↓	black 16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
•	blue	31	32	33	34	35	36	37	38	39	40	41	42	43	44
	green		45	46	47	48	49	50	51	52	53	54	55	56	57
	cyan			58	59	60	61	62	63	64	65	66	67	68	69
	red				70	71	72	73	74	75	76	77	78	79	80
m	agenta					81	82	83	84	85	86	87	88	89	90
	brown						91	92	93	94	95	96	97	98	99
	white							100	101	102	103	104	105	106	107
	gray								108	109	110	111	112	113	114
lig	ht blue									115	116	117	118	119	120
ligh	t green										121	122	123	124	125
	ht cyan											126	127	128	129
li	ght red												130	131	132
light m	agenta													133	134
	yellow														135

NOTE: To select a value that will cause blinking between colors A and B, find the number at the intersection of row A and column B.

DITHER COMBINATIONS FOR DEB PALETTES 0-3

Color combinations 136-255 have been pre-assigned to allow you easy access to dithering effects while using the standard palettes. The following table describes the available combinations.



NOTE: To select a value that combines colors A and B to create a new color, find the number at the intersection of row A and column B.

REMARKS

In the text-on-graphics overlay modes, all graphics statements except GET and PUT use page 128 (the DEB page). GET and PUT use the active page only. There is no way to GET or PUT an entire overlayed screen; you can only work with the active page.

In the graphics-on-graphics overlay modes, all graphics statements including GET and PUT use the active page only.

In all DEB modes, tiling with the PAINT command requires a 4-byte string rather than the 1-byte used in standard modes.

EXAMPLES

The following program demonstrates the PAL-ETTE USING statement to change the color combinations so that each color and its high intensity version are in consecutive positions in the palette.

```
50 SCREEN 102
                            '16 color graphics
 60 CLS:KEY OFF
                            'clear screen
 70 PALETTE
                            'use default palette
 80 DIM A%(256)
                            'array for PALETTE
 85 REM
                              USING statement
 90 J=0
100 FOR I = 0 TO 7
                            'load up the array
110 A\%(J) = 1:A\%(J + 1) = 1 + 8
120 J=J+2
130 NEXTI
140 LOCATE 2.2
150 FORI = 97 TO 112
                            'print 15 characters in
155 REM
                              15 colors
160 COLOR I-96,0:PRINT CHR$(I);
170 NEXTI
180 LOCATE 22,2
190 INPUT "Hit < CR > to change the colors", A$
195 REM
            Reprogram the entire palette
200 PALETTE USING A%(0)
210 LOCATE 22,2
220 INPUT "Hit < CR > to change the colors", A$
230 PALETTE
                            'use default palette
240 GOTO 180
```

The following example draws 3 interlocking circles in 16-color graphics mode and fills each separate section with various colors.

10	SCREEN 102	'set 16 color graphics
15	REM	mode
20	CLS:KEY OFF	'clear screen and turn
25	REM	functions keys off
30	COLOR ,,1	'use palette 1
35	REM	•
40	CIRCLE(320,200),100,15	'draw circle 1
50	CIRCLE(270,150),100,15	'draw circle 2
60	CIRCLE (370,150),100,15	'draw circle 3
70	PAINT (320,200),13,15	'fill with palette
75	REM	position 13
80	PAINT (269,150),12,15	'fill with palette
85	REM	position 12
90	PAINT (371,150),11,15	'fill with palette
95	REM	position 11
100	PAINT (320,250),10,15	'fill with palette
	REM	position 10
110	PAINT (320,100),9,15	'fill with palette
115	REM	position 9
120	PAINT (220,150),8,15	'fill with palette
	REM	position 8
130	PAINT (420,150),7,15	'fill with palette
	REM	position 7
140	FOR I = 7 TO 13	'loop thru the used
	REM	palette positions
150	PALETTE I,135 + RND*12	O 'use a random
155	REM	dithered color for
157	REM	palette position
160	FOR A = 1 TO 100:NEXT A	'wait awhile
	NEXTI	
	IF LEN(INKEY\$) = 0 THE	
	REM	'check for keypress
	SCREEN 0,0,0	return to normal
200	END	

The following program uses a tiling pattern to fill in a circle.

```
30 SCREEN 102
                             'set 16 color graphics
 40 CLS
                             'clear screen
                             'turn function keys off
 50 KEY OFF
 60 CIRCLE(320,200), 100.1 'draw a circle
            do the tiling to fill the circle
 80 PAINT(320,200), CHR$(&HCC) + CHR$
    (&H3C) + CHRS(&HC) + CHRS(&H3).1
 90 IF(LEN (INKEY$)) = 0
    THEN 90
                             'check for keypress
100 SCREEN 0.0.0
                             'return to normal
110 END
```

This program draws a small circle and cycles through all the available color combinations for the standard palette.

30	SCREEN 101	'16 color 640×200
35	REM	graphics
40	CLS	'clear screen
50	CIRCLE (320,100),100,1	'draw a circle
60	PAINT (320,100),1,1	'fill the circle with
65	REM	palette position 1
70	FOR J = 0 TO 255	'use all color
75	REM	combinations
80	PALETTE 1,J	'change the palette
85	REM	position color
90	FOR A = 1 TO 500:NEXT	A 'wait a bit
100	IF(LEN (INKEY\$))<>0	
	THEN 120	'check for
105	REM	keypress
110	NEXT J	
120	SCREEN 0,0,0	return to normal
130	END	

This program shows 3 ways in which a box can be drawn with palette position 2 and filled with palette position 14.

	SCREEN 102 CLS:KEY OFF	'16 color graphics 'clear screen
	DRAW "c2r50u50l50	ciear Screen
60	d50br2bu2p14.2"	'draw a box
70		
	REM	and fill it in
75	REM	
80	LINE (270,100)-	
	(320,150),2,B	'draw a box
90	LINE (271,101)-	
	(3 21,151),14,BF	'fill it in
100	REM	
110	LINE (220,150)-	
	(270,200),2,B	'draw a box
120	PAINT (221,151),14,2	'fill it in
130	IF LEN(INKEY\$) = O THE	N 130
140	SCREEN 0.0.0	
	END	
.50		

The following example draws a wheel with the number of spokes you specify, using random colors. Then it uses the PALETTE statement to cycle through the standard colors.

```
10 SCREEN 102: CLS:
     KEY OFF
                            'set 16 color
 15 REM
                              graphics
 20 INPUT "Number of spokes on wheel - ";N
 30 ANGLE = 360/N
                            'calculate # of angles
 40 RADIANS = ANGLE /57.29578
 50 CLS
                            'clear screen
 60 FOR X = 1 TO N
                            'do the real work
 70 FOR Y = X TO N
 80 SX = SIN(X * RADIANS) * 195 + 320
 90 SY = SIN(Y * RADIANS) * 195 + 320
100 CX = COS(X * RADIANS) * 150 + 200
110 CY = COS(Y * RADIANS) * 150 + 200
120 LINE (SY, CY)- (SX, CX),
    INT(RND*(15) + 1)
                            'draw line with
125 REM
                              random color
130 NEXT Y, X
140 FOR I = 1 TO 1000
150 FOR J = 1 TO 15
160 FORK = 1 TO 15
170 PALETTE K.J
                            'change palette
180 IF (LEN(INKEY$)) <> 0
    THEN 220
                            'check for
185 REM
                              keypress
190 NEXT K
200 NEXT J
210 NEXTI
220 SCREEN 0,0,0
                            'return to normal
230 END
```

This program demonstrates overlay mode by drawing a box on the DEB screen and a circle on the VDC screen. It then cycles through the blinking color combinations on the DEB and the standard colors on the VDC.

30	SCREEN 106	'8 color graphics on
35	REM	graphics overlay
40	CLS:KEY OFF	'clear screen
50	CIRCLE (320,200),100,1	'draw a circle
55	REM	on the VDC screen
60	PAINT (320,200),	
	CHR\$(1) + CHR\$(1),1	fill the circle with
65	REM	palette position 1
70	LOCATE 23,2;	-
75	PRINT "The circle is on th	e VDC screen";
80	SCREEN,,128	'set the active page
85	REM	to the DEB screen
90	LOCATE 24,2;	
95	PRINT "The box is on the I	DEB screen'';
100	LINE (250, 50)-	
	(390, 350), 5, BF	'draws a box on
105	REM	the DEB screen
110	FOR J=0 TO 135	'use all color
115	REM	combinations
120	SCREEN,,O:PALETTE	
	0,J-1 MOD 15	'change the palette
125	REM	position color on VDC
130	SCREEN,,128:PALETTE	-
	5,J	'change the palette
135	REM	position color on DEB
140	FOR A = 1 TO 500:NEXT A	A 'wait a bit
150	IF (LEN(INKEY\$)) <>0	
	THEN 170	'check for
	REM	keypress
160	NEXT J	
170	SCREEN 0,0,0	return to normal
180	END	

The following program takes two color numbers as input and finds their position in the dither and blinking tables and makes colored boxes in each of the color effects.

```
40 SCREEN 101
                             '16 Color 640 × 200
 45 REM
                              graphics mode
 50 CLS:KEY OFF
                             'clear screen
 60 REM Input the two colors and do range checking
 70 LOCATE 2.2:INPUT "Enter Color 1 (0-15)" .C1
 80 IF C1 > 15 OR C1 < 0 THEN GOTO 70
 90 LOCATE 3.2:INPUT "Enter Color 2 (0-15)".C2
100 IF C2 > 15 OR C2 < 0 THEN GOTO 90
110 IF C1 = C2 THEN INPUT "Colors must be different
       hit <CR>".AS:CLS:GOTO 70
120 REM Set one color to high and one to low to
125 REM determine the position in the respective
130 REM tables
140 IF C1 < C2 THEN LOW = C1:HIGH = C2
               ELSE LOW = C2:HIGH = C1
150 REM Blinking is the sum of 16-I as I ranges
155 REM from 0 to the lower of the two colors
160 REM then adding the higher of the two colors
170 ROWMIN = 0
180 FOR I = 0 TO LOW
190 ROWMIN = ROWMIN + (16-1)
200 NEXTI
210 BLINKCOL = ROWMIN + (HIGH-LOW-1)
220 LOCATE 22,1
230 PRINT "Blinking Number is ";BLINKCOL;
240 REM Dithering is 136 plus the sum of I + 1
245 REM as I ranges from 1 to the higher of the
250 REM two colors plus the lower color.
260 ROWMIN = 0
270 FOR I = 1 TO HIGH
280 ROWMIN = ROWMIN + (I-1)
290 NEXTI
295 REM example continued on next page
```

- 300 DITHERCOL = ROWMIN + 136 + LOW
- 310 LOCATE 22,42
- 320 PRINT "Dithered Color Number is "; DITHERCOL
- 330 REM Set palette position 1 equal to the
- 335 REM result of the blinking color
- 340 REM and palette position 2 equal to the
- 345 REM result of the dithering color
- 350 PALETTE 1,BLINKCOL
- **360 PALETTE 2, DITHERCOL**
- 370 REM draw a box with the blinking and
- 375 REM dithered color effects.
- 380 LINE (100,50)-(210,150),1,BF 390 LINE (420,50)-(530,150),2,BF
- 400 GOTO 70

The following program shows a box containing a circle and how the GET statement and the PUT statement work with the DEB. The GET array takes four times as much storage as it does in non-DEB graphics.

```
40 DIM PIC%(3000)
                             'GET arrav
 50 KEY OFF
                             'turn off function keys
 60 SCREEN 102
                             'set 16 color graphics
 70 FOR X = 1 TO 15
 80 CLS
                             'clear screen
 90 CIRCLE (100,100),50,1
                             'draw circle
100 LINE (49.50)-(151.150), 15-X.B
105 REM draw a box around the circle
110 PAINT (100.100).X.1
                             'fill the circle
120 GET (49,50)-(151,150),
     PIC%
                             'get the graphics
125 REM
                               image
130 FOR J = 1 TO 200 STEP 50
140 FOR I = 0 TO 50 STEP 10
150 PUT (RND*537 + 1.RND*297 + 1), PIC%, PSET
155 REM
                             'put it randomly on the
157 REM
                               screen
160 IF LEN(INKEY$) () 0
    THEN 210
                             'see if key
165 REM
                               pressed
170 NEXTI
180 NEXTJ
190 NEXT X
200 GOTO 70
210 SCREEN 0.0.0
                             'return to normal
220 END
```

The following program shows the use of a variety of DEB features. It includes a setup procedure to help you adjust your monitor for best viewing of DEB effects

```
1100 REM Display Enhancement Board
1200 REM Monitor Setup Program
1300 REM
1400 SCREEN 0,0,0
1500 KEY OFF:CLS
1600 REM
1700 REM The following is a way to easily center
1800 REM the title text
1900 A$="AT&T PC-6300"
1910 LOCATE 1,(80-LEN(A$))/2;
1920 PRINTAS
                            'Center text
2000 AS="DISPLAY ENHANCEMENT BOARD"
2010 LOCATE 2.(80-LEN(A$))/2:PRINT A$
2100 A$="MONITOR SETUP PROGRAM"
2110 LOCATE 3,(80-LEN(A$))/2:PRINT A$
2200 LOCATE 10,1:INPUT "Enter Monitor type
             ('MONO' or 'COLOR')";M$
2300 IF LEFT$(M$.1) = "M" OR LEFT$(M$.1) = "m"
            THEN GOTO 2900
2400 IF LEFT$(M$,1) = "C" OR LEFT$(M$,1) = "c"
            THEN GOTO 5000
2500 PRINT
2510 PRINT CHR$(7);"Can not use "';M$;" as a monitor
           type"
2600 FOR A = 1 TO 3000:NEXT A
2700 GOTO 2200
2800 REM
2900 REM Monochrome Monitor Setup
3000 REM
3100 DIM PAL(16)
3200 SCREEN 102: CLS
3300 FOR A=0 TO 15
3310 READ PAL(A):PALETTE A.PAL(A)
```

```
3320 NEXTA
                             'setup grav levels
3400 FOR A=0 TO 15
3500 LINE (A*40,40)-(40 + A*40,140), A,BF
3510 REM
                             'draw shaded areas
3600 LINE (A*40.240)- (40 + A*40.340), 15-A.BF
3610 REM draw inverted shaded areas
3700 NEXT A
3800 COLOR 15
                             'use high intensity white
3810 REM
                              for text
3900 LOCATE 1.20:
3910 PRINT "Adjust to get a complete shade scale"
4000 LOCATE 11.26:
4010 PRINT "Dark <-----> Light"
4100 LOCATE 14.25:
4110 PRINT "Light <-----> Dark"
4200 LOCATE 25.30:
4210 PRINT "(Hit any key to exit)":
4300 AS = INKEYS:IF LEN(AS) = 0 THEN 4300
4310 REM 'wait for any key to be pressed
4400 SCREENO
4500 REM
4600 REM The data below is the palette for
4700 REM shades of green
4800 DATA 0,8,1,9,4,12,5,13, 2,10,3,11,6,14,7,15
4900 END
5000 REM
5100 REM Color Monitor Setup
5200 REM
5300 SCREEN 102:CLS
5400 COLOR ..2
                             'select standard color
5410 REM
                              palette
5500 FOR A = 0 TO 7
5600 LINE (A*40.0)-(40 + A*40.199). A.BF
5610 REM draw colored filled boxes
5700 LINE (A*40,202)-(40+A*40,400), A+8,BF
5800 NEXTA
5900 COLOR 15
                             'use high intensity white
5910 REM
                               for text
6000 LOCATE 6,45: PRINT "Low intensity Colors"
6100 LOCATE 20.45: PRINT "High Intensity Colors"
```

6200 LOCATE 12,45;
6210 PRINT "Adjust Contrast and Brightness"
6300 LOCATE 13,45: PRINT "Controls to display 16"
6400 LOCATE 14,45: PRINT "different colors"
6500 LOCATE 25,50: PRINT "(Hit any key to exit)";
6600 A\$ = INKEY\$:IF LEN(A\$) = 0 THEN 6600
6610 REM wait for a key to be pressed
6700 SCREEN 0 'reset the screen mode
6800 END

The following program shows a text screen scrolling on top of a graphics screen.

```
20 SCREEN 104
                          'set text on graphics
 22 REM
                            mode
 25 CLS: KEY OFF
 30 N=15:ANGLE = 360/N 'calculate # of angles
 40 RADIANS = ANGLE /57.29578
                          'clear screen
 50 CLS
 60 FOR X = 1 TO N
                          'do the real work
 70 FORY = X TO N
 80 SX = SIN(X * RADIANS) * 195 + 320
 90 SY = SIN(Y * RADIANS) * 195 + 320
100 CX = COS(X * RADIANS) * 150 + 200
110 CY = COS(Y * RADIANS) * 150 + 200
120 LINE (SY,CY)-(SX,CX), INT(RND*(7) + 1)
125 REM draw line with random color
130 NEXT Y,X
140 FOR I = 1 TO 1000
150 X=RND*14+1
155 Y=RND*50+1
157 COLOR,,(RND*30),(RND*15)
159 GOSUB 270
                          'print text on VDC
160 X=RND*17+1
161 Y=RND*50+1
163 COLOR,,0,(RND*31+1)
165 GOSUB 270
                          'print text on VDC
167 COLOR,,,0
                          'change palette
170 LOCATE 24.1
180 FORK = 1 TO 7
190 PALETTE K,RND*135+1 'change palette
200 PRINT
                          'scroll text
210 IF(LEN(INKEY$)) <> 0 THEN 240
215 REM
                          check for keypress
220 NEXTK
230 NEXTI
240 SCREEN 0,0,0
                         return to normal
250 END
```

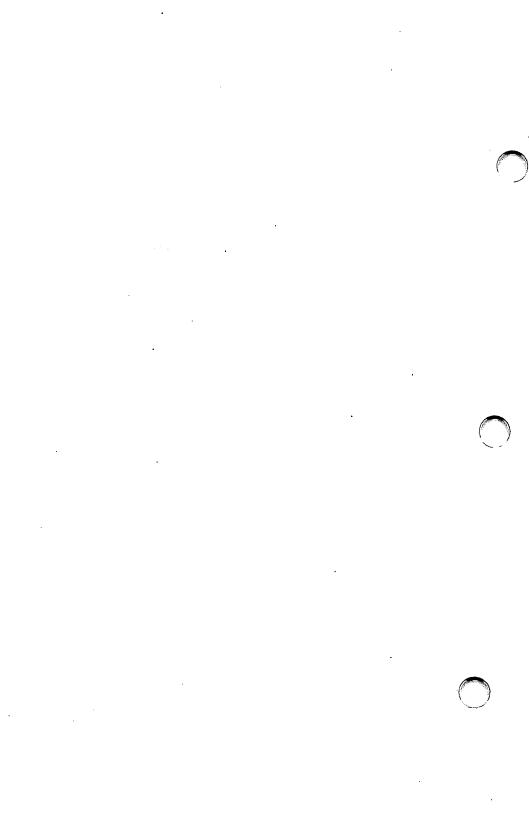
```
260 REM sub to display a box of text
270 LOCATE X,Y: PRINT CHR$(201);
280 FOR I = 1 TO 29:PRINT CHR$(205): :NEXT I
290 PRINT CHR$(187);
300 LOCATE X + 1.Y:
305 PRINT CHR$(186) + "This box is on the VDC
              screen" + CHR$(186);
310 LOCATE X + 2.Y:
315 PRINT CHR$(186) + "This is more text"
              + CHR$(186):
320 LOCATE X + 3, Y;
325 PRINT CHR$(186) + "This is the last line of text"
              + CHR$(186):
330 LOCATE X + 4, Y: PRINT CHR$(200);
340 FOR I = 1 TO 29: PRINT CHR$(205)::NEXT I
350 PRINT CHR$(188);
360 RETURN
```

ξ ... •

4

Programming the LUT

- Overview
- 16-Color Graphics LUT Programming
- Overlay Modes LUT Programming



OVERVIEW

This chapter describes programming the DEB look-up table (LUT). By programming the LUT yourself, you can create color patterns that are not available when you use standard palettes.

You need not read this chapter if you do not want to use this extended functionality.

The hardware uses the LUT to translate the contents of video memory into graphics effects. In the standard palettes, GWBASIC programs the LUT for you and thereby provides the pre-assigned color combinations and effects described in previous chapters.

To program the LUT directly, you select Palette 4 in the COLOR statement. Palette 4, also called the "LUT palette," has a minimum of 256 positions. The contents of each palette position is an integer value between 0 and 15. These values map into the LUT locations on the DEB. The 256 locations on the DEB collectively determine the color and special effects displayed when you specify a particular palette position in a graphics statement. The color and special effect for each pixel on the screen are determined by:

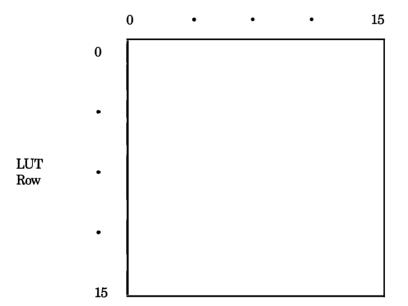
- the palette position you specify
- the values in the LUT
- the active mode

There are some differences in the way the LUT is structured for 16-color graphics modes and overlay modes. This chapter describes LUT operation for 16-color graphics modes and overlay modes separately.

16-COLOR GRAPHICS LUT PROGRAMMING

In these modes the LUT can be viewed as a twodimensional array (16×16). Each location contains one of the standard 16 colors.

Palette Position



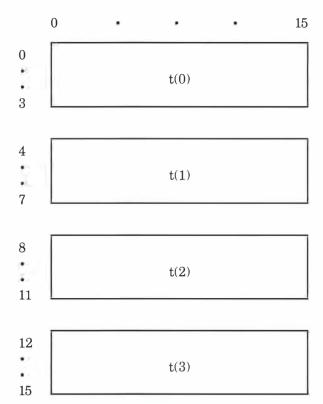
The locations in the LUT are numbered consecutively from left to right and top to bottom. Thus, location 17 corresponds to Row 1, palette position 1. This correspondence is used with both the PALETTE and PALETTE USING statements. To set location 17 to color 1 (blue) you would either use:

PALETTE 17,1

 \mathbf{or}

INTARRAY (17) = 1 PALETTE USING INTARRAY (0) In the 16-color graphics mode, the LUT is divided into four "time states." At any one time, only one quarter of the LUT determines the display on the screen.

Palette Position



LUT

Row

The hardware cycles through the LUT every second, so each quarter of the LUT is active for ¼ of each second. The cycling mechanism produces blinking. The following examples show the details of how you can produce several different blinking effects by setting different values in the LUT.

In this example, the graphics statements specify palette position 7 and the LUT is set up as shown. Pixels are displayed as a solid red color. In the first ¼ second, the DEB displays the color in the first quarter of the LUT, which in this case is red. In the second, third, and fourth ¼ seconds, the DEB displays the color in the second, third, and fourth quarters of the LUT, respectively. In this example, the DEB keeps finding the color value for red, so what you see on the screen is a solid (non-blinking) red color.

	LUT		Palette Position	
	Row	0	7	15
t0	0 • • 3		red	
t1	4 • 7		red	
t2	8 • • 11		red	
t3	12 • • 15		red	

Non-Blinking Color

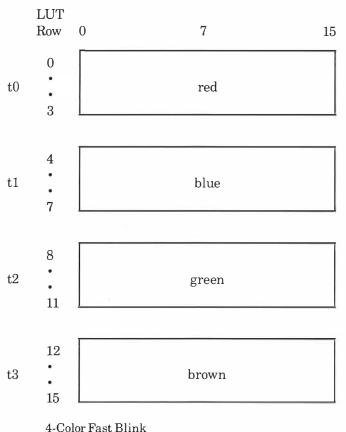
In this example, any item displayed on the screen with palette position 7 blinks between red and blue. For the first two ¼ seconds, the DEB picks up the color value for red from the first and second quarters of the LUT. For the second two ¼ seconds, the DEB obtains the color value of blue from the LUT. The net effect is a slow blink between red and blue.

Palette Position

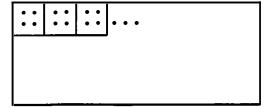
	LUT Row	0	7	15
t0	0 • • 3		red	
t1	4 • • 7		red	
t2	8 • • 11		blue	
t3	12 • • 15		blue	
	Slow	Blink		

In this example, any item displayed using palette position 7 blinks rapidly between red, blue, green, and brown.

Palette Position



For dithering colors, the DEB uses a scheme similar to the blinking scheme. Dithering is accomplished by manipulating groups of 4 adjacent pixels. The screen is divided into blocks of 4 pixels.



Each of the 4 time states is divided into four dither states that determine the dithering effect. The rows of the time state blocks correspond to the 4-pixel blocks on the screen in the following way:

	Time		Palette Position	
LUT Row	Block Row	0		15
0 1 2 3	0 1 2 3		t(0)	
4 5 6 7	0 1 2 3		t(1)	
8 9 10 11	0 1 2 3		t(2)	
12 13 14 15	0 1 2 3		t(3)	

The pixels in the pixel blocks are so close together that our eyes cannot perceive them as separate. If each of the pixels in a pixel block is a different color, our eyes perceive the pixel block as one color—a combination of the color of the individual pixels. If the adjacent pixels are the same color, our eyes see just that one color.

		Palette Position	
Time Block	0	7	15
Row	0	red	
	1	red	
t(0)	2 3	red	
	3	red	
	0	red	
		red	
t(1)	1 2 3	red	
	3	red	
	0	red	
		red	i
t(2)	1 2 3	red	
	3	red	
	0	red	<u> </u>
		red	
t(3)	2	red	
(0)	1 2 3	red	
	-		

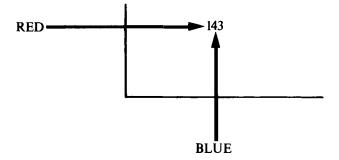
red

red

red red

"Solid" Dither showing correspondence between pixel positions in a pixel block and time state rows

Remember the table of "pre-assigned" dithered colors in Chapter 3. To combine colors, you check the table for the color number for a particular dither effect. For example, you would choose this number to produce a dither between red and blue.



If you want to program the LUT to dither red and blue together, the LUT would look like this:

blue	red	blue	red
blue	red	blue	red

Palette Position

Block Row 0	7	15
0 1 2 3	blue red blue red	

	0	blue
	1	red
t(1)	2	blue
	3	red

Time

t(0)

	0	blue
	1	red
t(2)	2	blue
	3	red

	0	blue
	1	red
t(3)	2	blue
	3	red

2-Color Dither

You can set up the LUT to dither two, three, or four colors together. $\,$

Palette Position

	Time Block Row 0	7	15
t(0)	0 1 2 3	red blue green brown	
t(1)	0 1 2 3	red blue green brown	
t(2)	0 1 2 3	red blue green brown	
t(3)	0 1 2 3	red blue green brown	

4-Color Dither

The following examples show the actual LUT values for each of the previous cases of blinking and dithering.

Palette Position

	LUT Row 0	7	15
t(0)	0 1 2 3	4 (red) 4 4 4	
t(1)	4 5 6 7	4 4 4 4	
t(2)	8 9 10 11	4 4 4 4	
t(3)	12 13 14 15	4 4 4 4	

Palette Position 7 programmed for Non-Blinking Red

Palette Position

	LUT Row 0	7	15
t(0)	$\begin{bmatrix}0\\1\\2\\3\end{bmatrix}$	4 (red) 4 4 4	
t(1)	4 5 6 7	4 4 4 4	
t(2)	$\begin{bmatrix} 8 \\ 9 \\ 10 \\ 11 \end{bmatrix}$	1 (blue) 1 1 1	
t(3)	12 13 14 15	1 1 1 1	

 $Palette\ Position\ 7\ programmed\ to\ blink\ slowly\ between\ red\ and\ blue.$

Palette Position

	LUT Row 0	7	15
t(0)	0 1 2 3	4 (red) 4 4 4	
t(1)	4 5 6 7	1 (blue) 1 1 1	
t(2)	8 9 10 11	2 (green) 2 2 2 2	
t(3)	12 13 14	6 (brown) 6 6	

4-Color Fast Blink

14 15

Palette Position LUT Row 0 4 (red) $\begin{array}{c} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \end{array}$ t(0) 5 6 7 $\overline{4}$ t(1) 9 t(2) t(3)

Solid Red Dither

	1.1100	Palette Position	
	LUT Row 0	7	15
t(0)	0 1 2 3	1 (blue) 4 (red) 1 (blue) 4 (red)	
t(1)	4 5 6 7	1 4 1 4	
t(2)	8 9 10 11	1 4 1 4	
t(3)	12 13 14 15	1 4 1 4	

2-Color Dither: Red and Blue

Palette Position

	LUT Row	0	7	15
t(0)	0 1 2 3		4 (red) 2 (green) 1 (blue) 6 (brown)	
t(1	4 5 6 7		4 2 1 6	3
t(2	8 9 10 11		4 2 1 6	
t(3)	12 13 14 15		4 2 1 6	

4-Color Dither Between Red, Green, Blue, and Brown

The following is an example that combines blinking and dithering:

Palette Position

	LUT Row	0	7	15
t(0)	0 1 2 3		1 (blue) 4 (red) 1	
t(1)	4 5 6 7		1 4 1 4	
t(2)	8 9 10 11		2 (green) 6 (brown) 2 6	
t(3)	12 13 14 15		2 6 2 6	!

The following table of values can be used to program the LUT for normal 16-color graphics.

Palette Position

```
LUT
       Row
                 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
          0
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          1
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
t(o)
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          2
          3
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          4
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          5
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
t(1)
          6
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          7
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          8
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          9
t(2)
         10
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
        11
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
        12
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
        13
                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
t(3)
        14
        15
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
```

Non-Blinking Standard Colors

Note that palette position 7 in the first two time states has been programmed to show white and in the second two time states to show red.

Palette Position

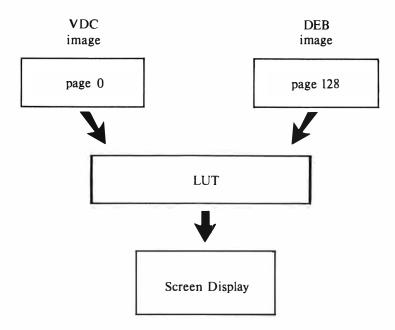
```
LUT
       Row
                0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          0
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          1
t(0)
          2
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          3
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          4
          5
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
t(1)
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          6
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
          7
          8
                0, 1, 2, 3, 4, 5, 6, 4, 8, 9, 10, 11, 12, 13, 14, 15,
          9
                0, 1, 2, 3, 4, 5, 6, 4, 8, 9, 10, 11, 12, 13, 14, 15,
t(2)
                0, 1, 2, 3, 4, 5, 6, 4, 8, 9, 10, 11, 12, 13, 14, 15,
        10
        11
                0, 1, 2, 3, 4, 5, 6, 4, 8, 9, 10, 11, 12, 13, 14, 15,
        12
                0, 1, 2, 3, 4, 5, 6, 4, 8, 9, 10, 11, 12, 13, 14, 15,
        13
                0, 1, 2, 3, 4, 5, 6, 4, 8, 9, 10, 11, 12, 13, 14, 15,
t(3)
        14
                0, 1, 2, 3, 4, 5, 6, 4, 8, 9, 10, 11, 12, 13, 14, 15,
        15
                0, 1, 2, 3, 4, 5, 6, 4, 8, 9, 10, 11, 12, 13, 14, 15,
```

LUT for Blinking Between White and Red in Palette Position 7

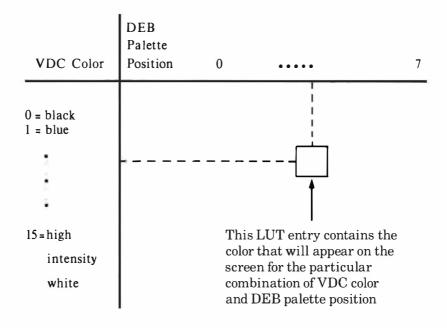
OVERLAY MODES LUT PROGRAMMING

When the LUT is used in the overlay modes it can be viewed as a two-dimensional array with 8 columns and 32 rows. The column values are DEB palette positions. The row values are VDC color values.

In overlay modes, there are 2 separately controlled images: the VDC image and the DEB image. The 2 images are combined on the display screen. Each pixel on the screen has 2 values associated with it: the VDC color and the DEB palette position. The LUT is used to resolve contention between the 2 values associated with each pixel.



The LUT for overlay modes looks like this:



As in the 16-color graphics modes, the locations in the LUT are numbered consecutively from left to right and top to bottom. For example, location 17 corresponds to Row 2, Palette Position 0.

In the overlay modes, as in the 16-color graphics mode, the LUT is divided into time states that control blinking effects. However, in the overlay modes, the LUT is only divided into two time states. Half of the LUT determines what is being displayed at any time. The top half is used for the first ½ of each second and the bottom half is used for the second ½ of each second.

Using the overlay modes, you create blinking by making the values in the top half of the table different from the corresponding values in the bottom half of the table.

DEB Palette Position

LUT Row	0	•	•	•	7
0 • • 15			t(0)		
16 • • 31			t(1)		

.

VDC

The following example shows the LUT values for standard Palette 2 of an overlay mode. The LUT is programmed so that the DEB image is displayed only if the VDC color is 0 (black). If the VDC requests any other color, then that color is displayed no matter what the DEB requests. This has the effect of overlaying the VDC image "on top" of the DEB image.

DEB Palette Position

	Color								
	Values	0	1	2	3	4	5	6	7
	0	0,	1,	2,	3,	4,	5,	6,	7,
	1	1,	1,	1,	1,	1,	1,	1,	1,
	2	2,	2,	2,	2,	2,	2,	2,	2,
	3	3,	3,	3,	3,	3,	3,	3,	3,
t(0)	4	4,	4,	4,	4,	4,	4,	4,	4,
	5	5,	5,	5,	5,	5,	5,	5,	5,
	6	6,	6,	6,	6,	6,	6,	6,	6,
	7	7,	7,	7,	7,	7,	7,	7,	7,
	8	8,	8,	8,	8,	8,	8,	8,	8,
	9	9,	9,	9,	9,	9,	9,	9,	9,
	10	10,	10,	10,	10,	10,	10,	10,	10,
	11	11,	11,	11,	11,	11,	11,	11,	11,
	12	12,	12,	12,	12,	12,	12,	12,	12,
	13	13,	13,	13,	13,	13,	13,	13,	13,
	14	14,	14,	14,	14,	14,	14,	14,	14,
	15	15,	15,	15,	15,	15,	15,	15,	15,

DEB Palette Position

VDC Color Values 0 1 2 3 4 5 6 7

3, 4, 6, 0 1, 2, 5, 7, 1, 1, 1, 1, 1 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2 2, 2, 3, 3 3, 3, 3, 3, 3, 3, 3, t(1)4 4, 4, 4, 4, 4, 4, 4, 4, 5 5, 5, 5, 5, 5, 5, 5, 5, 6 6, 6, 6, 6, 6, 6, 6, 6, 7 7, 7, 7, 7, 7, 7, 7, 7, 8 8, 8, 8, 8, 8, 8, 8, 9, 9, 9 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 10, 10, 10, 10 11, 11, 11, 11, 11, 11, 11, 11, 11 12 12, 12, 12, 12, 12, 12, 12, 12, 13 13, 13, 13, 13, 13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14 15 15, 15, 15, 15, 15, 15, 15, 15,

In this example, the standard Palette 2 is modified so that position 2 is a blinking between blue (color 1) and red (color 4).

DEB Palette Position

	VDC								
	Color								
	Values	0	1	2	3	4	5	6	7
	1								
	0	0,	1,	1,	3,	4,	5,	6,	7,
	1	1,	1,	1,	1,	1,	1,	1,	1,
	2	2,	2,	2,	2,	2,	2,	2,	2,
	3	3,	3,	3,	3,	3,	3,	3,	3,
t(0)	4	4,	4,	4,	4,	4,	4,	4,	4,
	5	5,	5,	5,	5,	5,	5,	5,	5,
	6	6,	6,	6,	6,	6,	6,	6,	6,
	7	7,	7,	7,	7,	7,	7,	7,	7,
	8	8,	8,	8,	8,	8,	8,	8,	8,
	9	9,	9,	9,	9,	9,	9,	9,	9,
	10	10,	10,	10,	10,	10,	10,	10,	10,
	11	11,	11,	11,	11,	11,	11,	11,	11,
	12	12,	12,	12,	12,	12,	12,	12,	12,
	13	13,	13,	13,	13,	13,	13,	13,	13,
	14	14,	14,	14,	14,	14,	14,	14,	14,
	15	15.	15.	15,	15.	15.	15.	15.	15.

DEB Palette Position

VDC

Color Values 6 7 0 1 2 3 4 5 0 4, 3, 4, 5, 6, 7, 1, 1 1, 1, 1, 1, 1, 1, 1, 1, 2 2, 2, 2, 2, 2, 2, 2, 3 3, 3, 3, 3, 3, 3, 3, t(1) 4 4, 4, 4, 4, 4, 4, 4, 5 5, 5, 5, 5, 5, 5, 5, 6 6, 6, 6, 6, 6, 6, 6, 7 7, 7, 7, 7, 7, 7, 7, 8 8, 8, 8, 8, 8, 8, 8, 9 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 10, 10, 10, 10 11 11, 11, 11, 11, 11, 11, 11, 11, 12, 12, 12, 12, 12, 12, 12, 12, 12 13 13, 13, 13, 13, 13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14 15 15, 15, 15, 15, 15, 15, 15, 15,

In this example, values in the LUT cause the DEB's output to take precedence over the VDC's output. The VDC's output is only displayed when you specify DEB palette position 0 in a graphics statement.

DEB Palette Positions

	VDC	
	Color	
	Values	$0\ 1\ 2\ 3\ 4\ 5\ 6\ 7$
		 -
	0	0, 1, 2, 3, 4, 5, 6, 7,
	1	1, 1, 2, 3, 4, 5, 6, 7,
	2	2, 1, 2, 3, 4, 5, 6, 7,
	3	3, 1, 2, 3, 4, 5, 6, 7,
	4	4, 1, 2, 3, 4, 5, 6, 7,
	5	5, 1, 2, 3, 4, 5, 6, 7,
	6	6, 1, 2, 3, 4, 5, 6, 7,
t(o)	7	7, 1, 2, 3, 4, 5, 6, 7,
	8	8, 1, 2, 3, 4, 5, 6, 7,
	9	9, 1, 2, 3, 4, 5, 6, 7,
	10	10, 1, 2, 3, 4, 5, 6, 7,
	11	11, 1, 2, 3, 4, 5, 6, 7,
	12	12, 1, 2, 3, 4, 5, 6, 7,
	13	13, 1, 2, 3, 4, 5, 6, 7,
	14	14, 1, 2, 3, 4, 5, 6, 7,
	15	15. 1. 2. 3. 4. 5. 6. 7.

VDC Color Values 0 1 2 3 4 5 6 7 0, 1, 2, 3, 4, 5, 6, 7, 1 **0**, 1, 2, 3, 4, 5, 6, 7, 1 2 2, 1, 2, 3, 4, 5, 6, 7, 3 3, 1, 2, 3, 4, 5, 6, 7, 4, 1, 2, 3, 4, 5, 6, 7, 4 5 **5,** 1, 2, 3, 4, 5, 6, 7, t(1) 6 **6**, 1, 2, 3, 4, 5, 6, 7, 7 **7**, 1, 2, 3, 4, 5, 6, 7, 8 8, 1, 2, 3, 4, 5, 6, 7, 9, 1, 2, 3, 4, 5, 6, 7, 9 **10,** 1, 2, 3, 4, 5, 6, 7, 10 11, 1, 2, 3, 4, 5, 6, 7, 11 12 **12,** 1, 2, 3, 4, 5, 6, 7, 13 **13,** 1, 2, 3, 4, 5, 6, 7, 14, 1, 2, 3, 4, 5, 6, 7, 14

15, 1, 2, 3, 4, 5, 6, 7,

15

DEB Palette Positions

VDC Color

0

15

The following LUT entirely blocks out VDC output:

Values 0 1 2 3 4 5 6 7 0, 1, 2, 3, 4, 5, 6, 7,0, 1, 2, 3, 4, 5, 6, 7,0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,0, 1, 2, 3, 4, 5, 6, 7,

0, 1, 2, 3, 4, 5, 6, 7,

DEB Palette Positions

1 2 3 4 t(0)5 0, 1, 2, 3, 4, 5, 6, 7,6 7 0, 1, 2, 3, 4, 5, 6, 7,0, 1, 2, 3, 4, 5, 6, 7,8 0, 1, 2, 3, 4, 5, 6, 7,9 0, 1, 2, 3, 4, 5, 6, 7,10 0, 1, 2, 3, 4, 5, 6, 7,11 0, 1, 2, 3, 4, 5, 6, 7,12 13 0, 1, 2, 3, 4, 5, 6, 7,0, 1, 2, 3, 4, 5, 6, 7,14

DEB Palette Positions

Color 0 1 2 3 4 5 6 7 Values 0, 1, 2, 3, 4, 5, 6, 7,0 0, 1, 2, 3, 4, 5, 6, 7,1 2 0, 1, 2, 3, 4, 5, 6, 7,3 0, 1, 2, 3, 4, 5, 6, 7,

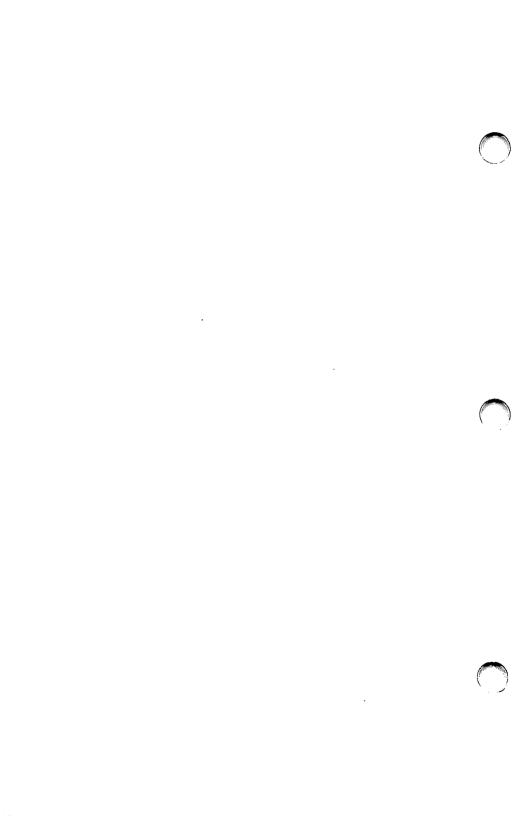
VDC

15

4 0, 1, 2, 3, 4, 5, 6, 7,t(1) 5 0, 1, 2, 3, 4, 5, 6, 7,6 0, 1, 2, 3, 4, 5, 6, 7,7 0, 1, 2, 3, 4, 5, 6, 7,8 0, 1, 2, 3, 4, 5, 6, 7,9 0, 1, 2, 3, 4, 5, 6, 7,0, 1, 2, 3, 4, 5, 6, 7,10 0, 1, 2, 3, 4, 5, 6, 7,11 12 0, 1, 2, 3, 4, 5, 6, 7,0, 1, 2, 3, 4, 5, 6, 7,13 14 0, 1, 2, 3, 4, 5, 6, 7,0, 1, 2, 3, 4, 5, 6, 7,

APPENDICES

A	Tables
В	Advanced Features
C	Conversion of Programs to GWBASIC
D	Error Codes and Error Messages



A

Tables

- Hexadecimal Conversion Tables
- ASCII Codes
- Extended Codes
- Hexadecimal to Decimal Conversion Tables
- Derived Functions



Hexadecimal Conversion Table

Control characters (codes 0 to 31)

Printed characters (codes 32 to 127)

Dec	Hex	Char	Function	Dec	He	Char	Dec	Hex	Char	Dec	He	Char
0	00	NUL	Null-no character, no action	32	20	SPace	64	40	@	96	60	
1	01	SOH	Start of Heading*	33	21	!	65	41	Α	97	16	a
2	02	STX	Start of Text*	34	22		66	42	В	98	62	b
3	03	ETX	End of Text (Block mode-ETX/ACK)	35	23	#	67	43	C	99	63	c
4	04	EOT	End of Transmission*	36	24	S	68	44	D	100	64	d
5	05	ENQ	Enquiry*	37	25	%	69	45	E	101	65	e
6	06	ACK	Acknowledge (Block mode-ETX/ACK)	38	26	&	70	46	F	102	66	f
7	07	BEL	Bell-ring or beep	39	27		71	47	G	103	67	g
8	08	BS	Backspace-Move back one character	40	28	(72	48	Η	104	68	h
9	09	HT	Horizontal Tab-skip to next stop	41	29)	73	49	I	105	69	i
10	0A	LF	Line Feed-Move down one line	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	Vertical Tab*	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	Form Feed-Move to new page	44	2C	,	76	4C	L	108	6C	1
13	0D	CR	Carriage Return-Go to start of line	45	2D		77	4D	M	109	6D	m
14	0E	SO	Shift Out*	46	2E		78	4E	N	110	6E	n
15	0F	SI	Shift ln*	47	2F	1	79	4F	O	111	6F	0
16	10	DLE	Data Link Escape*	48	30	0	80	50	P	112	70	p
17	11	DC1	Device Control 1-X-ON (CTRLQ)	49	31	1	81	51	Q	113	71	q
18	12	DC2	Device Control 2*	50	32	2	82	52	R	114	72	r
19	13	DC3	Device Control 3-X-OFF (CTRLS)	51	33	3	83	53	S	115	73	S
20	14	DC4	Device Control 4*	52	34	4	84	54	T	116	74	t
21	15	NAK	Negative Acknowledge*	53	35	5	85	55	U	117	75	u
22	16	SYN	Synchronous Idle*	54	36	6	86	56	V	118	76	v
23	17	ETB	End of Transmission Block*	55	37	7	87	57	W	119	77	w
24	18	CAN	Cancel*	56	38	8	88	58	X	120	78	x
25	19	EM	End of Medium*	57	39	9	89	59	Y	121	79	У
26	1 A	SUB	Substitute*	58	3A	:	90	5A	Z	122	7A	Z
27	1B	ESC	Escape (Normally calls command menu)	59	3B	;	91	5B	[123	7B	{
28	1C	FS	File Separator*	60	3C	<	92	5C	\	124	7C	
29	1D	GS	Group Separator*	61	3D	=	93	5D]	125	7D	}
30	1E	RS	Record Separator*	62	3E		94	5E	^	126		
31	1F	US	Unit Separator*	63	3F	?	95	5F	_	127	7 F	DELete

Functions marked * are unused or archaic

DEL (127) is actually a control character

ASCII		Control	ASCII	- (
value	Character	character	value	Character
000	(null)	NUL	032	(space)
001	\odot	SOH	033	1
002	•	STX	034	ñ
003	♥	ETX	035	#
004	•	EOT	036	\$
005	•	ENQ	037	%
006	•	ACK	038	&
007	(beep)	BEL	039	•
800		BS	040	(
009	(tab)	HT	041)
010	(line feed)	LF	042	*
011	(home)	VT	043	+
012	(form feed)	FF	044	
013	(carriage return)	CR	045	9 1
014		SO	046	
015	☼	SI	047	/
016	•	DLE	048	0
017		DC1	049	1
018	‡	DC2	050	2
019	!!	DC3	051	3
020	प	DC4	052	4
021	9	NAK	053	5
022	_	SYN	054	6
023	<u>‡</u>	ETB	055	7
024	†	CAN	056	8
025	1	EM	057	9
026	→	SUB	058	3
027	←	ESC	059	;
028	(cursor right)	FS	060	<
029	(cursor left)	GS	061	=
030	(cursor up)	RS	062	>
031	(cursor down)	US	063	?

ASCII		ASCII	
value	Character	value	Character
064	@	096	*
065	Α	097	а
066	В	098	b
067	С	099	С
068	D	100	d
069	E	101	е
070	F	102	f
071	G	103	g
072	H	104	h
073	1	105	i
074	J	106	j
075	K	107	k
076	L	108	1
077	M	109	m
078	N	110	n
079	0	111	0
080	Р	112	р
081	Q	113	þ
082	R	114	r
083	S	115	S
084	Т	116	t
085	U	117	u
086	V	118	V
087	W	119	w
088	X	120	×
089	Y	121	У
090	Z	122	Z
091	[123	{
092	1	124	1
093]	125	}
094 095	^	126	~
095	_	127	

ASCII value	Character	ASCII value	Character
128	Ç	160	á
129	ü	161	í
130	é	162	ó
131	â	163	ú
132	ä	164	ñ
133	à	165	Ñ
134	å	166	<u>a</u>
135	¢	167	ō
136	ê	168	Ċ
137	ë	169	
138	è	170	\neg
139	ï	171	1/2
140	î	172	1/4
141	ì	173	İ
142	Ä	174	((
143	Á	175	>>
144	É	176	3886
145	æ	177	***
146	Æ	178	***
147	ô	179	1
148	ö	180	\dashv
149	ò	181	=
150	û	182	-41
151	ù	183	⊐
152	ÿ Ö	184	=
153		185	긤
154	Ü	186	II
155	¢	187	╗
156	£	188	긛
157	¥	189	_11
158	Pt	190	=
159	f	191	\neg

ASCII		ASCII	
value	Character	value	Character
192	L	224	α
193	_	225	β
194	T	226	Γ
195	†	227	π
196	_	228	Σ
197	+	229	σ
198	F	230	μ
199	 	231	T
200	L	232	Φ
201	F	233	- 0-
202	北	234	Ω
203	T F	235	δ
204	ŀ	236	∞
205	=	237	Ø
206	1 }	238	ϵ
207	±	239	Π
208	ш.	240	=
209	-	241	<u>+</u>
210	T	242	2
211	<u>L</u>	243	≤
212	L	244	ſ
213	F	245	J
214	г	246	÷
215	#	247	≈
216	+	248	0
217	٦	249	•
218	Γ	250	•
219		251	$\sqrt{}$
220	_	252	n
221	I_	253	2
222		254	
223	_	255	(blank 'FF')

EXTENDED CODES

An extended code is returned by the INKEY\$ system function for certain keys or key combinations that cannot be represented in standard ASCII code. A null character (ASCII code 00) is returned as the first character of a two-character string. If a two-character string is received by INKEY\$, then you should go back and examine the second character to determine the actual key pressed. Usually, but not always, this second code is the scan code of the primary key that was pressed. The ASCII codes (in decimal) for this second character, and the associated key(s) are listed below.

EXTENDED CODES

Second Code Meaning (decimal)

3	(null character) NULL
15	(shift tab)
16-25	ALT-Q,W,E,R,T,Y,U,I,O,P
30-38	ALT-A,S,D,F,G,H,J,K,L
44-50	ALT-Z,X,C,V,B,N,M
59-68	function keys F1 through F10
	(when disabled as soft keys)
71	HOME
72	Cursor Up
73	PGUP
75	Cursor Left
77	Cursor Right
79	END
80	Cursor Down
81	PGDN
82	INS
83	DEL
84-93	F11-F20 (SHIFT-F1 through F10)
94-103	F21-F30 (CTRL-F1 through F10)
104-113	F31-F40 (ALT-F1 through F10)
114	CTRL-PRTSC
115	CTRL-Cursor Left (Previous Word)
116	CTRL-Cursor Right (Next Word)
117	CTRL-END
118	CTRL-PGDN
119	CTRL-HOME
120-131	ALT-1,2,3,4,5,6,7,8,9,0,-,=
132	CTRL-PGUP

HEXADECIMAL TO DECIMAL CONVERSION TABLES

		BYTE			BYTE		
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
В	45056	В	2816	В	176	В	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	208	D	13
\mathbf{E}	57344	\mathbf{E}	3584	\mathbf{E}	224	\mathbf{E}	14
\mathbf{F}	61440	F	3840	\mathbf{F}	240	F	15

BINARY TO HEXADECIMAL CONVERSION TABLES

The following table shows the decimal (base 10), binary (base 2), and hex (base 16) representations for the numbers 0 to 16.

DECIMAL	BINARY	HEX
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	Α
11	1011	В
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

DERIVED FUNCTIONS

You can define a derived function in your program by use of a DEF FN statement to avoid coding the formula each time you need it.

Functions that are not intrinsic to GWBASIC may be calculated as follows.

Function	GWBASIC Equivalent
SECANT	SEC(x)=1/COS(x) when x<>1.570796
COSECANT	CSC(x)=1/SIN(x) when $x <> 0$
COTANGENT	$CO'\Gamma(x) = 1/\Gamma A N(x)$ when $x <> 0$
INVERSE SINE	ARCSIN(x) = ATN(x/SQR(1-x*x))
INVERSE COSINE	ARCCOS(x)=1.570796-ATN(x/SQR(1-x*x)) when $ABS(x) < 1$
INVERSE SECANT	ARCSEC(x) = ATN(SQR(x*x-1)) + SGN(SGN(x)-1)*1.570796 when $ABS(x) > 1$
INVERSE COSECANT	ARCCSC(x)=A'TN(1/SQR(x*x-1)) when ABS(x) > 1+(SGN(x)-1)*1.570796
INVERSE COTANGENT	ARCCOT(x) = 1.570796 - ATN(x)
HYPERBOLIC SINE	SINH(x) = (EXP(x)-EXP(-x))/2
HYPERBOLIC COSINE	COSH(x) = (EXP(x) + EXP(-x))/2
HYPERBOLIC TANGENT	TANH(x) = (EXP(x)-EXP(-x))/(EXP(x)+EXP(-x))
HYPERBOLIC SECANT HYPERBOLIC	SECH(x) = 2/(EXP(x) + EXP(-x))
COSECANT HYPERBOLIC	CSCH(x) = 2/(EXP(x)-EXP(-x)) when $x <> 0$
COTANGENT	$\begin{aligned} & \text{COTFI}(\mathbf{x}) \!=\! (\text{EXP}(\mathbf{x}) + \text{EXP}(\!\!\cdot\!\!\!\mathbf{x})) / (\text{EXP}(\mathbf{x}) \!\!\cdot\!\! \text{EXP}(\!\!\cdot\!\!\mathbf{x})) \\ & \text{when } \mathbf{x} \! < \! > \! 0 \end{aligned}$
INVERSE HYPERBOLIC SINE	ARCSINH(x) = LOG(x + SQR(x*x+1))
INVERSE HYPERBOLIC COSINE	ARCCOSII(x) = LOG(x + SQR(x*x-1)) when x > 1
INVERSE HYPERBOLIC	•
TANGENT	$\begin{split} & ARCTANH(x) \!=\! LOG((1+x)/(1-x))/2 \text{ when} \\ & ABS(x) \!<\! 1 \end{split}$

DERIVED FUNCTIONS

Function GWBASIC Equivalent

INVERSE HYPERBOLIC

SECANT ARCSECH(x)=LOG((SQR(1-x*x)+1)/x) when

0 < x < = 1

INVERSE HYPERBOLIC

COSECANT ARCCSCH(x)=LOG((SGN(x)*SQR(x*x+1)+1/x)

when x > 0

INVERSE HYPERBOLIC

COTANGENT ARCCOTH(x)=LOG((x+1)/(x-1))/2

when ABS(x) > 1

LOGARITHM TO BASE'a' LOGA(x)=LOG(x)/LOG(a) when a>0 and x>0

Note: Both 'x' and 'a' can be any numeric constant, variable, array element, function or expression. Any values of 'x' or 'a' that would cause error messages are noted.

,

B

Advanced Features

- Memory Allocation
- Internal Representation
- Calling Subroutines
- Event Trapping

MEMORY ALLOCATION

Memory space must be set aside for an assembly language subroutine before it can be loaded. To do so, use the /M: option on the GWBASIC command (refer to the GWBASIC command in the Command Reference section). The /M: option sets the highest memory location to be used by GWBASIC.

In addition to the GWBASIC code area, GWBASIC uses up to 64K of memory beginning at its data (DS) segment.

If extra stack space is needed when an assembly language subroutine is called, you can save the GWBASIC stack and set up a new stack for use by the assembly language subroutine. The GWBASIC stack must be restored, however, before you return from the subroutine

The assembly language subroutine can be loaded into memory in several ways, the most simple being to use the BLOAD commmand (see the BLOAD Command in the Reference section). Also, you could execute a program that exits but stays resident, and then run GWBASIC. As a third choice, the assembled instructions could be stored in DATA statements and POKEd directly into memory.

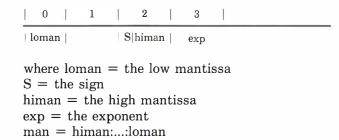
The following guidelines must be observed if you choose to BLOAD, or read and poke, an EXE file into memory:

- Make sure the subroutines do not contain any long references, address offsets that exceed 64K or that take the user out of the code segment. These long references require handling by the EXE loader.
- Skip over the first 512 bytes (the header) of the linker's output file (EXE), then read in the rest of the file.

INTERNAL REPRESENTATION

The following section describes the internal representation of numbers in GWBASIC.

Single Precision-24 bit mantissa



• If exp=0, then number=0.

• If exp <>0, then the mantissa is normalized and

That is, in single precision (hex notation - bytes low to high)

$$00000080 = 1.5$$

 $00008080 = -.5$

Double Precision - 56 bit mantissa

0	J	1	2	3	4		5		6		7	
lomar	n					1		S	hima	an	exp	1

CALLING SUBROUTINES

CALL STATEMENT

The CALL statement is the recommended way of calling machine language programs with GWBASIC. It is preferable to the USR function unless you are running programs that already contain USR functions.

The syntax of the CALL statement is:

Syntax

CALL numvar [(variable [,variable]...)]

numvar

contains the offset into the current segment that is the starting point in memory of the subroutine being called. The list of variables indicates variables or constants, separated by commas, that are to be passed to the subroutine as arguments. The current segment is either the default, or that which has been defined by a DEF SEG statement.

Invoking the CALL statement causes the following to occur:

- For each variable specified in the statement, the two-byte offset of the variable's location within the GWBASIC segment is pushed onto the stack.
- The GWBASIC return address code segment (CS) and offset (IP) are pushed onto the Stack.
- Control is transferred to the machine language routine using the segment address, which is given in the last DEF SEG statement and the offset given in numvar.

The following diagrams illustrate the state of the stack at the time the CALL statement is executed and the condition of the stack during execution of the called subroutine, respectively.

high addresses		argument 0	SP+4+(2#n) Each argument is a 2-byte		
		•	pointer into memory		
	1	•			
	c				
s	0	argument n-1			
\mathbf{t}	u		SP+6		
a	n	argument n			
c	t		SP+4		
k	e	return segment address			
	r		SP+2		
	\downarrow	return offset			
		SP	stack pointer		
	low		(SP register		
ado	lresses		contents)		

Stack Layout When CALL Statement is Activated

After the CALL statement has been activated, the subroutine has control. Arguments may be referenced by moving the stack pointer (SP) to the base pointer (BP) and adding a positive offset to BP.

```
high
addresses argument 0
          argument 1
                                   Absent if any argument is
                                   referenced within a nested
    1
                                   procedure
       c
s
       o argument n
t
       u
       t return segment address Absent in local procedure
       r return offset
                                               stack pointer
                                   (SP register
          local variables
                                   contents)
          (data pushed on
          stack)
          This space may be
          used during pro-
                                   Stack pointer may change
          cedure execution
                                   during procedure
                                   execution
  low
addresses
```

Stack Layout During Execution of a CALL Statement

Observe the following rules when coding a subroutine:

- The called routine must preserve segment registers DS, ES, SS, and BP. If interrupts are disabled in the routine, they must be enabled before exiting. The stack must be cleaned up on exit.
- The called program must know the number and length of the arguments passed. The following routine shows an easy way to reference arguments:

```
push BP
mov BP,SP
add BP, (2*number of arguments)+4
```

Then:

```
argument 0 is at BP argument 1 is at BP-2 argument n is at BP-2*n (number of arguments=n+1)
```

 Variables may be allocated either in the Code Segment or on the stack. Be careful not to modify the return segment and offset stored on the stack.

- The called subroutine must clean up the stack.
 A preferred way to do this is to perform a RET n statement (where n is two times the number of arguments in the argument list) to adjust the stack to the start of the calling sequence.
- Values are returned to GWBASIC by including in the argument list the name of the variable that will receive the result.
- If the argument is a string, the argument's offset points to 3 bytes which, as a unit, are called the string descriptor. Byte 0 of the string descriptor contains the length of the string (0 to 255). Bytes 1 and 2, respectively, are the lower and upper 8 bits of the string starting address in string space.

If the argument is a string literal in the program, the string descriptor will point to program text. Be careful not to alter or destroy your program this way. To avoid unpredictable results, Concatenate a null string to the string literal in the program. For example, use:

20 A\$="BASIC"+""

This will force the string literal to be copied into string space. Then the string may be modified without affecting the program.

- The contents of a string may be altered by user routines, but the descriptor must not be changed. Do not write past the end-of-string. GWBASIC cannot correctly manipulate strings if their lengths are modified by external routines.
- Data areas needed by the routine must be allocated either in the CODE segment of the user routine or on the stack. It is not possible to declare a separate data area in the user assembler routine.

Example

100 DEF SEG = &H8000 110 FOO = &H7FA 120 CALL FOO (A,B\$,C)

Line 100 sets the segment to 8000 Hex. The value of variable FOO is added into the address as the low word after the DEF SEG value is left shifted 4 bits, i.e., multiplied by 16. (This is a function of the microprocessor, not of GWBASIC.) Here, FOO is set to &H7FA, so that the call to FOO will execute the subroutine at location 8000:7FA Hex (absolute address 8007FA Hex).

The following sequence in 8086 assembly language demonstrates access to the arguments passed. The returned result is stored in the variable C.

PUSH	BP	;Set up pointer
MOV	BP,SP	
ADD	BP,(4+2*3)	
MOV	BX,[BP-2]	;Get address of B\$
MOV	CL,[BX]	Get length of B\$
MOV	DX,1[BX]	;Get addr of B\$ text
	•	
	•	
	•	
MOV	SI,[BP]	;Get address of 'A'
MOV	DI[BP-4]	;Get pointer to 'C'
MOVS	WORD	;Store variable 'A'
POP	BP	;Restore pointer.
RET	6	:Restore stack

Note: The called program must know the variable type for the numeric arguments passed. In the previous example, the instruction:

MOVS WORD

will copy only two bytes. This is fine if variables A and C are integer. You would have to copy four bytes if the variables were single precision format and copy 8 bytes if they were double precision.

CALLS STATEMENT

The CALLS statement should be used to access subroutines that were written using MS-FORTRAN calling conventions. CALLS works just like CALL, except that with CALLS the arguments are passed as segmented addresses, rather than as unsegmented addresses.

Because MS-FORTRAN routines need to know the segment value for each argument passed, the segment is pushed and then the offset is also pushed. For each argument, four bytes are pushed rather than 2, as in the CALL statement. Therefore, if your assembler routine uses the CALLS statement, n in the RET statement is two times the number of arguments +2.

USR FUNCTION

Although using the CALL statement is the recommended way of calling assembly language routines, the USR function is also available for this purpose. This ensures compatibility with older programs that contain USR functions.

The format of the USR function is:

Syntax

USR [n] (argument)

 \mathbf{n}

is a digit from 0 to 9. It specifies which user routine is being called. If n is omitted, USR0 is assumed.

argument

is any numeric or string expression.

A DEF SEG statement must be executed prior to a USR function call to ensure that the code segment points to the subroutine being called. The segment address given in the DEF SEG statement determines the starting segment of the subroutine.

For each USR function, a corresponding DEF USR statement must be executed to define the USR function call offset. This offset and the currently active DEF SEG address determine the starting address of the subroutine.

When the USR function call is made, register AL contains a value that specifies the type of x argument that was given. The value in AL may be one of the following:

2	Two-byte integer (two's complement)
3	String
4	Single precision floating-point number
8	Double precision floating-point number

If the argument is a number, the BX register points to the Floating-Point Accumulator (FAC) where the argument is stored.

If the argument is an integer:

FAC-2 contains the upper 8 bits of the argument.

FAC-3 contains the lower 8 bits of the argument.

If the argument is a single precision floatingpoint number:

FAC-2 contains the middle 8 bits of mantissa. FAC-3 contains the lowest 8 bits of mantissa.

If the argument is a double precision floatingpoint number:

FAC-7 through FAC-4 contain four more bytes of mantissa (FAC-7 contains the lowest 8 bits).

If the argument is a string, the DX register points to 3 bytes which, as a unit, are called the string descriptor. Byte 0 of the string descriptor contains the length of the string (0 to 255 characters). Bytes 1 and 2, respectively, are the lower and upper 8 bits of the string starting address in the GWBASIC data segment. If the argument is a string literal in the program, the string descriptor will point to program text. Be careful not to alter or destroy the program this way.

Usually, the value returned by a USR function is the same type (integer, string, single precision, or double precision) as the argument that was passed to it.

GWBASIC has extended the USR function interface to allow calls to MAKINT and FRCINT. This allows access to these routines without giving their absolute addresses. The address ES:BP is used as an indirect far pointer to the routines FRCINT and MAKINT.

To call FRCINT from a USR routine use

CALL DWORD ES:[BP]

To call MAKINT from a USR routine use

CALL DWORD ES:[BP+4]

Example

110 DEF USRO=&H8000 115 'Assumes user gave /M:32767 120 X=5 130 Y=USRO(X) 140 PRINT Y

The type (numeric or string) of the variable receiving the function call must be consistent with that of the argument passed.

EVENT TRAPPING

Event trapping allows a program to transfer control to a specific program line when a certain event occurs. Control is transferred as if a GOSUB statement had been executed to the trap routine starting at the specified line number. The trap routine, after servicing the event, executes a RETURN statement that causes the program to resume execution at the place where it was when the event trap occurred.

The events that can be trapped are receipt of characters from a communication port (ON COM (n) GOSUB), detection of certain keystrokes (ON KEY (n) GOSUB), time passage (ON TIMER (n) GOSUB), or emptying of the background music queue (ON PLAY (n) GOSUB).

Event trapping is controlled by the following statements:

Syntax 1 (to turn on trapping)

Syntax 2 (to turn off trapping)

Syntax 3 (to temporarily turn off trapping)

Remarks

COM (n)

where n is the number (1 through 4) of the communications channel.

Typically, the COM trap routine will read an entire message from the COM port before returning. We do not recommend using the COM trap for single character messages because at high baud rates the overhead of trapping and reading for each character may allow the interrupt buffer for COM to overflow.

KEY (n)

where n is a trappable key number. Trappable keys are numbered 1 through 20.

Note that KEY(n) ON is not the same statement as KEY ON. KEY(n) ON sets an event trap for the specified key. KEY ON displays the values of all the function keys on the twenty-fifth line of the screen.

When GWBASIC is in direct mode function keys maintain their standard meanings.

When a key is trapped, that occurrence of the key is destroyed. Therefore, you cannot subsequently use the INPUT or INKEY\$ statements to find out which key caused the trap. So if you wish to assign different functions to particular keys, you must set up a different subroutine for each key, rather than assigning the various functions within a single subroutine.

TIMER

The ON TIMER(n) GOSUB statement (where n is a numeric expression representing a number of seconds since the previous midnight) can be used to perform background tasks at defined intervals.

PLAY

The ON PLAY(n) GOSUB statement (where n is a number of notes left in the music buffer) is used to retrieve more notes from the background music queue, to permit continuous background music during program execution.

THE ON GOSUB STATEMENT

The ON GOSUB statement sets up a line number for the specified event trap. The format is:

ON {COM(n) | KEY(n) | TIMER(n) | PLAY(n)} GOSUB linenum

A linenum of zero disables trapping for that event.

When an event is ON and if a non-zero line number has been specified in the ON GOSUB statement, every time GWBASIC starts a new statement it will check to see if the specified event has occurred (e.g., a COM character has come in). When an event is OFF, no trapping takes place, and the event is not remembered even if it takes place.

When an event is STOPped, no trapping takes place, but the occurrence of that event is remembered so that an immediate trap will take place when the associated event ON statement is executed.

When a trap is made for a particular event, the trap automatically causes a STOP on that event, so recursive traps can never occur. A return from the trap routine automatically executes an ON statement unless an explicit OFF has been performed inside the trap routine.

Note that once an error trap takes place, all trapping is automatically disabled. In addition, event trapping will never occur when GWBASIC is not executing a program.

THE RETURN STATEMENT

When an event trap is in effect, a GOSUB statement will be executed as soon as the specified event occurs. For example, the statement:

ON KEY(10) GOSUB 1000

specifies that the program go to line 1000 as soon as Function Key F10 is pressed. If a simple RETURN statement is executed at the end of this subroutine, program control will return to the statement following the one where the trap occurred. When the RETURN statement is executed, its corresponding GOSUB return address is cancelled.

GWBASIC includes the RETURN linenum enhancement, which lets processing resume at a definable line. Normally, the program returns to the statement immediately following the GOSUB statement when the RETURN statement is encountered. However, RETURN linenum enables you to specify another line. If not used with care, however, this capability may cause problems. Assume, for example, that your program contains:

10 ON KEY(10) GOSUB 1000
20 FOR I = 1 TO 10
30 PRINT I
40 NEXT I
50 REM NEXT PROGRAM LINE
200 REM PROGRAM RESUMES HERE
1000 'FIRST LINE OF SUBROUTINE
1050 RETURN 200

If the Function Key F10 is pressed while the FOR/NEXT loop is executing, the subroutine will be performed, but program control will return to line 200 instead of completing the FOR/NEXT loop. The original GOSUB entry will be cancelled by the RETURN statement, and any other GOSUB, WHILE, or FOR that was active at the time of the trap will remain active. The current FOR context will also remain active, and a FOR without NEXT error may result.

• ı

C

Conversion of Programs to GWBASIC

- Introduction
- String Dimensioning
- MAT Functions
- Multiple Assignments
- Multiple Statements
- PEEKs and POKEs
- IF...THEN...[ELSE...]
- File I/O
- Graphics
- Sounding the Bell

INTRODUCTION

GWBASIC bears a similarity to many BASICs. Your personal computer will support programs written for an extensive variety of microcomputers. For programs written in a BASIC other than GWBASIC, some minor adjustments may be necessary before running them. This appendix highlights some specific areas to examine when converting programs.

STRING DIMENSIONING

LENGTH OF STRINGS

GWBASIC strings are of variable lengths. Therefore, all statements that declare the length of strings should be deleted. For example, in a statement which dimensions a string array for 'J' elements of lengths 'I' such as:

DIM A\$(I,J)

the conversion for GWBASIC would be:

DIM AS(J)

SUBSTRINGS

In GWBASIC the following functions are used to take substrings of strings:

LEFT\$ MID\$ RIGHT\$

Other forms, such as:

AS(I) (to access the Ith character in A\$) or **AS(I,J)** (to take a substring of A\$ from position I to J) should be changed as follows:

Other BASICs GWBASIC

XS=AS(I) = XS=MIDS(AS,I,1)XS=AS(I,J) = XS=MIDS(AS,I,J-I+1)

If the substring reference is on the left side of an assignment and X\$ is used to replace characters in A\$, then the conversion should be carried out as follows:

Other BASICs GWBASIC

A\$(I)=X\$ = MID\$(A\$,I,1)=X\$ A\$(I,J)=X\$ = MID\$(A\$,I,J-I+1)=X\$

CONCATENATION

GWBASIC uses a plus (+) sign to denote string concatenation. Other BASICs use a comma (,) or an ampersand (&) which should be altered accordingly.

MAT FUNCTIONS

Any programs which use the MAT function (available in some BASICs) must be rewritten incorporating FOR...NEXT loops before they will run properly.

MULTIPLE ASSIGNMENTS

Some BASICs allow the following syntax:

10 LET D=E=0

to set D and E equal to zero. GWBASIC interprets the second equal sign as a logical operator and sets D equal to -1 if E was equal to 0. This statement should therefore be broken up into two assignment statements as follows:

10 D=0:E=0

MULTIPLE STATEMENTS

Multiple statements on a GWBASIC line must always be separated by colons (:), unlike some other BASICs which use a backslash (\) instead.

PEEKs AND POKEs

The execution of programs containing PEEK and POKE instructions may vary from machine to machine. It is therefore necessary to analyze the purpose of these instructions in other BASIC programs before translating the same functions into GWBASIC.

IF...THEN...[ELSE...]

Not all BASICs feature the optional ELSE clause which is performed in the event of a test proving false.

For example, a BASIC statement may originally be:

10 IF D=E THEN 30 20 PRINT "NOT EQUAL" : GOTO 40 30 PRINT "EQUAL" 40 REM CONTINUE

The above statement sequence will work correctly, but it may be optimized in GWBASIC as follows:

10 IF D=E THEN PRINT "EQUAL"
ELSE PRINT "NOT EQUAL"
20 REM CONTINUE

FILE I/O

In GWBASIC, the reading and writing of information to and from a disk file is achieved by opening the file to associate it with a particular file number, then selecting particular I/O statements that specify that file number. In some other BASICs, the I/O to disk is somewhat different. Refer to Chapter 4, "Disk File Handling," and to the Reference section under the OPEN statement, for fuller descriptions.

GRAPHICS

Drawing an image on the screen can vary from BASIC to BASIC. Refer to Chapter 5, "Graphics," and the Reference section for a description of the available graphic statements.

SOUNDING THE BELL

The PRINT CHR\$(7) is required for some BASICs to send an ASCII bell character. Under GWBASIC the BEEP, SOUND and PLAY statements can also be used to sound the bell.

ş . -

D

Error Codes and Error Messages

- Error Messages
- Error Codes

ERROR 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	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
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

NUMBER	MESSAGE
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 filename
66	Direct statement in file
67	Too many files
68	Device unavailable
69	Communication buffer overflow
70	Disk write protected
71	Disk not ready
72	Disk media error
74	Rename across disks
75	Path/file access error
76	Path not found

CODE NUMBER

NF	1	NEXT without FOR A NEXT statement has been encountered without a matching FOR
SN	2	Syntax error A line is encountered which includes an incorrect sequence of characters (misspelled keyword, incorrect punctuation, etc.). GWBASIC automatically enters edit mode at the line that caused the error.
RG	3	RETURN without GOSUB A RETURN statement is encountered for which there is no previous, unmatched GOSUB statement.
OD	4	Out of data A READ statement is executed when there are no DATA statements with unread data remaining in the program.
FC	5	Illegal function call A parameter that is out of range is passed to a numeric or string function. This FC error may also occur as the result of:
		 A negative or unreasonably large subscript. A negative or zero argument with LOG. A negative argument to SQR. A negative mantissa with a noninteger exponent. A call to a USR function for which the starting address has not yet been given. An improper argument to MID\$, LEFT\$, RIGHT\$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING\$, SPACE\$, INSTR, or ONGOTO.

MESSAGE

CODE	NUMBER	MESSAGE
OV	6	Overflow The result of a calculation is too large to be represented in GWBASIC number format. If underflow occurs, the result is zero and execution continues without an error.
OM	7	Out of memory A program is too big, or has too many loops, subroutines, variables; or has expressions that are too complicated to evaluate
UL	8	Undefined line number A nonexistent line is referenced in a GOTO, GOSUB, IFTHENELSE, or DELETE statement.
BS	9	Subscript out of range An array element is referenced either with a subscript that is outside the dimensions of the array or with the wrong number of subscripts.
DD	10	Duplicate Definition Two DIM statements are given for the same array; or a DIM statement is given for an array after the default dimension of 10 has been established for that array; or an OPTION BASE is given after an array has been dimensioned.
/0	11	Division by zero A division by zero is encountered in an expression; or, the value zero has been raised to a negative power. In the former case, the result is machine infinity (with the appropriate sign); in the latter case, the result is positive machine infinity.

CODE	NUMBER	MESSAGE
ID	12	Illegal direct A statement that is illegal in direct mode is entered as a direct mode command.
TM	13	Type mismatch A string variable name is assigned a numeric value or vice versa; a function that expects a numeric argument is given a string argument or vice versa.
●S	14	Out of string space String variables have caused GWBASIC to exceed the amount of free memory remain- ing. GWBASIC will allocate string space dynamically, until it runs out memory.
LS	15	String too long An attempt is made to create a string in excess of 255 characters.
ST	16	String formula too complex A string expression is too long or too complex to be processed. It should be broken into smaller expressions.
CN	17	Can't continue An attempt is made to continue a program that: 1. Has halted due to an error. 2. Has been modified during a break in execution. 3. Does not exist.
UF	18	Undefined user function A USR function is called before the function definition (DEF statement) is given.

CODE	NUMBER	MESSAGE
	19	No RESUME An error handling routine is entered but contains no RESUME statement.
	20	RESUME without error A RESUME statement is encountered before an error handling routine is entered.
	22	Missing operand An expression contains an operator with no operand following it.
	23	Line buffer overflow An attempt has been made to input a line that has too many characters.
	24	Device Timeout GWBASIC did not receive information from an I/O device within a predetermined amount of time.
	25	Device Fault In GWBASIC, will only occur when a fault status is returned from the Line Printer inter- face. Usually indicates a hardware error in the printer or interface card.
	26	FOR without NEXT A FOR statement was encountered without a matching NEXT.
	27	Out of paper The printer is out of paper or is not switched on. Insert paper, ensure power is switched on and continue.

CODE NUMBER MESSAGE

29 WHILE without WEND

A WHILE statement does not have a matching WEND.

30 WEND without WHILE

A WEND statement was encountered without a matching WHILE.

50 FIELD overflow

A FIELD statement is attempting to allocate more bytes than were specified for the record length of a random file.

51 Internal error

An internal malfunction has occurred in GWBASIC. Report the conditions under which the message appeared to the AT&T Information Systems Services Hotline.

52 Bad file number

A statement or command references a file with a file number that is not OPEN or is out of the range of file numbers specified at initialization.

53 File not found

A LOAD, KILL, or OPEN statement references a file that does not exist on the current disk.

54 Bad file mode

An attempt is made to use PUT, GET, or LOF with a sequential file, to LOAD a random file, or to execute an OPEN statement with a file mode other than I, O, or R.

55 File already open

A sequential output mode OPEN statement is issued for a file that is already open; or a KILL statement is given for a file that is open.

CODE NUMBER MESSAGE

57 Device I/O Error

An I/O error occurred on a peripheral device I/O operation. It is a fatal error; i.e., the operating system cannot recover from the error.

58 File already exists

The filename specified in a NAME statement is identical to a filename already in use on the disk.

61 Disk full

All disk storage space is in use.

62 Input past end

An INPUT statement is executed after all the data in the file has been INPUT, or for a null (empty) file. To avoid this error, use the EOF function to detect the end-of-file.

63 Bad record number

In a PUT or GET statement, the record number is either greater than the maximum allowed (32,767) or equal to zero.

64 Bad filename

An illegal form is used for the filename with a LOAD, SAVE, KILL, or OPEN statement (e.g., a filename with too many characters).

66 Direct statement in file

A direct statement is encountered while LOADing an ASCII-format file. The LOAD is terminated.

67 Too many files

An attempt is made to create a new file (using SAVE or OPEN) when all 255 directory entries are full.

CODE NUMBER

MESSAGE

68 Device unavailable

An attempt was made to open a file to a non-existent device. It may be that hardware did not exist to support the device, such as LPT2: or LPT3:, or was disabled by the user. This occurs if an OPEN "COM1:... statement is executed after the user has disabled RS232 support via the /C:0 switch directive on the command line.

69 Communication buffer overflow

Not enough space has been reserved for communications I/O. Several options are available:

- Increase the size of the COM receive buffer via the /C: switch.
- 2. Implement a "hand-shaking" protocol with the host/satellite such as XON/XOFF as demonstrated in the TTY programming example to turn transmit off long enough to catch up.
- Use a lower baud rate for transmit and receive.

70 Disk Write Protected

This is one of 3 "hard" disk errors returned from the disk controller. This occurs when an attempt is made to write to a disk that is write protected. Use an ON ERROR GOTO statement to detect this situation a d request user action. Other possible "hard" disk errors are:

71 Disk not ready

Occurs when the disk drive door is open or a disk is not in the drive. Again use an ON ERROR GOTO statement to recover.

CODE NUMBER

MESSAGE

72 Disk media error

Occurs when the FDC controller detects a hardware or media fault. This usually indicates damaged media. Copy any existing files to a new disk and reformat the damaged disk. FORMAT will flag the bad tracks and place them in a file "badtrack." The remainder of the disk is now usable.

74 Rename across disks

An attempt was made to rename a file to a new name that was declared to be on a disk other than the disk specified for the old name. The renaming operation is not performed.

75 Path/file access error

During an OPEN, MKDIR, CHDIR or RMDIR operation, MS-DOS was unable to make a correct Path to File connection. The operation is not completed.

76 Path not found

During an OPEN, MKDIR, CHDIR, or RMDIR operation, MS-DOS was unable to find the path specified. The operation is not completed.

•

Glossary

GLOSSARY

absolute coordinate form

In graphics, specifiying the location of a pixel with respect to the origin of the specified coordinate system.

access mode

A technique that is used to obtain a specific logical record from, or place a specific logical record into, a file.

active page

The screen buffer which has information written to it. It may be different from the visual page whose information is being displayed.

address

Location in storage.

addressable point

The technique of displaying a sequence of images so that you can see the objects moving on the screen (see the GET and PUT graphics

statements).

argument

A value that is passed from the main program to a function or a subroutine.

array

A collection of variables of the same type under one name. You can distinguish them by the value(s) of one or more subscripts.

2

array element An element of an array. It is a variable whose name coincides with the name of the array and is followed by one or more subscripts in parentheses. They specify the position of the array element within the array.

ASCII

American Standard Code for Information Interchange. A standard 8 bit code used for exchanging information among data processing systems and associated equipment.

aspect ratio

Determines the spacing of the horizontal, vertical, and diagonal points. The standard aspect ratio of 4/3 indicates that the horizontal axis of the screen is 4/3 as long as the vertical axis.

asynchronous

A method of transmitting data in which each transmitted character is preceded by a start bit and followed by a stop bit, thus permitting the interval between characters to vary.

background color

The color of the area which surrounds the subject. In particular, the color of the screen surrounding a character (character background color) or the color displayed when a graphics viewport is cleared (graphics background color).

baud The transmission rate which is in effect;

synonymous with signal events (usually bits) per

second.

boolean value A numeric value that is interpreted as "true" value (if it is not zero) or "false" (if it is zero).

bps Bits per second.

built-in function

"See intrinsic function."

call The branching or transfer of control to a

specified subroutine.

carriage return A character that causes the print or display return position to move to the first position on the next line. Entering CR when you finish typing a GWBASIC line, passes the line to

GWBASIC for processing.

character definition tag A special character placed at the end of a definition variable—It may be: % (integer definition variable), ! (single-precision variable), #

(double-precision variable), or \$ (string

variable).

clipping

The graphics statements use line clipping, i.e., lines that cross the screen or viewport are "clipped" at or cut off at the edges of the viewing area.

command level The GWBASIC is at command level when Ok appears on the screen, i.e., when it is waiting for the user to enter an immediate or program line.

comment

A statement used to document a program. In GWBASIC, a comment may be entered by REM or a single quote (') followed by the comment string. The single quote (') also allows the insertion of comments at the end of a GWBASIC line.

concatenation

The operation that joins two strings together.

constant

A fixed value or data item. A constant may be string or a numeric constant. In the latter case it may be an integer, a single-precision or a double-precision number.

coordinates

Numbers which identify a location on the screen. They may be text coordinates to identify a character or the cursor (expressed in terms of rows and columns) or graphics coordinates to identify a pixel (expressed as x and y Cartesian coordinates).

current directory The directory you are working on. You may change the current directory by the CHDIR command. Just after formatting a disk the ROOT directory is the current directory.

current line

The line you are working on, or the line you have just entered, or the line where an error has occurred

current point

See "last-referenced point."

current program The program currently in memory.

current viewport

The viewport you are working on. To change viewport you must use a VIEW statement.

cursor

A movable marker that is used to indicate a position on the screen. There are three types of cursor (see the LOCATE statement in the Reference section). The shape and blinkrate of the overwrite and user cursors are programmable. The user cursor is not visible at

initialization.

debug

To locate and correct mistakes in a program.

default Pertaining to a value or option that is assumed

when none is given.

delimeter A character that limits a string of characters

and therefore cannot be part of the string.

destination The variable to the left of the equal sign in an

assignment statement.

direct The ability to read or write information at any

access access location within a storage device.

direct line See "immediate line."

direct mode See "immediate mode."

directory The directory contains the names of files on the

disk, along with information that tells MS-DOS

where to find the file.

disk Is a generic term to specify either a hard-disk

or a diskette.

diskette A 51/4-inch mini floppy disk.

double This is the maximum precision GWBASIC can handle. If a number contains more than 7 digits it is a double-precision number.

drive Synonymous with disk drive. May be specified by A: (first diskette drive), B: (second diskette

drive), C: (hard-disk drive), etc.

dummy A fictitious parameter in a function or argument statement or command. A value must be entered, but it is ignored by GWBASIC.

edit To enter, modify, or delete a GWBASIC line.

end of file A "marker" immediately following the last (EOF) record of a file. It signals the end of the file.

error When an error occurs, the control of the trapping program may be automatically directed to a specified program line.

event When a certain event occurs, the control of the trapping program may be automatically directed to a specified program line. Events include: receipt of characters from a communication port, detection of certain keystrokes, time passage, emptying of

the background music queue.

expression An algorithm returning a single numeric value

(numeric, relational or logical expressions) or a string operation returning a string value

(string expression).

field In a record, a specific area used for a

particular type of data.

file A collection of records. The records of a file

may be accessed by GWBASIC sequentially (one after the other) or randomly (by record

number).

fixed- Enumerable elements in a file each of which has

length the same length. For example, random files have

fixed-length records.

filename Name assigned to a file.

file Unique file identifier. A file specification can

include a drive specifier (A:,B:,C:, etc.).

floppy A diskette.

specification

foreground The color of the character itself (character color foreground color), or the color used to draw

foreground color), or the color used to draw pictures when no color parameter is specified in

a graphics statement (graphics foreground

color).

full duplex A communication system permitting

simultaneous operation in both directions.

function An algorithm returning a single value. A

function can be a user or an intrinsic function. It can be called forth simply by stating its name, followed (in parentheses) by one or more "arguments" representing the values that the

function parameters are to assume.

function key A key to which the user can assign a special

meaning. Typing the key you may generate any character string. Some function keys may already be assigned by the system at

initialization.

graphics See "viewport." viewport

GWBASIC In this manual refers only to Microsoft

GWBASIC Version 2.02 as implemented on the

AT&T Personal Computer.

half duplex A communication system permitting operation

in either direction, but not simultaneously.

hard disk A rigid disk. In this manual, referring to a

51/4-inch Winchester-type disk.

immediate

line

A GWBASIC line which begins with a letter. It

is executed as soon as you press CR.

immediate mode This mode is used to immediately enter and

execute a GWBASIC line.

indirect line See "program line."

indirect mode See "program mode."

interrupt The suspension of a process, such as the

execution of a program, caused by an event external to that process, and performed in such

a way that the process can be resumed.

intrinsic function

A function that the user may call without defining it since it is an integral part of

GWBASIC (e.g., SIN(x)).

keyword

One of the predefined words of GWBASIC. It

is a reserved word.

lastreferenced point In graphics, the last-referenced point may be used for relative coordinates (see the STEP option in the LINE statement)

t option in the LINE statement).

line A GWBASIC line may begin with a line

number (if it is a program line) or with a letter

(if it is an immediate line). The line may

contain one or more GWBASIC statements or commands (separated by colons) and may be

up to 255 characters long.

line clipping See "clipping."

line folding
The continuation of a logical line on a

subsequent physical line, so that the line can be modified by insertion or deletion without losing any other characters on that line.

loop The repeated execution of a series of statements

for a fixed number of times.

machine The largest number that can be represented in

infinity internal format.

mantissa The numeral that is not the exponent in floating

point notation.

matrix See "array."

MS-DOS Microsoft-Disk Operating System.

nest	To embed a	subroutine	or block	of data	a into a	ı
	1	and the sale of	- L J - L -			

larger routine or block of data.

null A string with zero length, i.e., with no

characters in it. (It is represented as "".)

numeric expression

An expression whose evaluation returns a numeric value. This may be an integer, a single-precision or a double-precision value.

numeric keypad The section on the right of the keyboard dedicated to numbers, arithmetic symbols, cursor movement keys, and some control

characters.

numeric variable A simple variable or array element whose value is numeric; i.e., an integer, a single precision or a double precision, depending on the type

defined for the variable.

offset

The number of bytes from a starting point to some other point. For example, in GWBASIC a memory address may be given as an offset from the memory segment defined by the DEF SEG statement.

option switch One of the options in the GWBASIC command line switch specified with a slash (/) followed by a character or by a character and a colon.

overflow In an arithmetic operation, the generation of a

quantity beyond the capacity of a register or

location which is to receive the result.

overlay Programs too large for memory can be divided

into logical segments (or overlays).

parameter Value supplied to a command or language

statement that provides additional information

for the command or statement. Used

interchangeably with argument. An "actual parameter" is a value that is substituted for a "formal parameter" in a given procedure or

function when invoked.

pixel A graphics "point" addressable on the screen

by its coordinates (x,y). Also, the bits which

contain the information for that point.

port An access channel for data entry or exit.

program mode This mode is used to enter into memory a

GWBASIC program line. To tell GWBASIC the line you are entering is part of a program, you begin the line with a line number. A program line is stored in memory when you press CR, but it is not executed. The lines are stored in memory in line number sequence to form a GWBASIC program. To execute the program

press RUN CR

Message displayed on the screen to request the prompt

user to do a specific action.

record A group of one or more consecutive fields on a

related subject. For example, an employee's

payroll record. A file is a collection of

records.

reset To reload an operating system from disk into

memory.

redirection You can redirect your GWBASIC input and

> output by the GWBASIC command. Standard input can be redirected to any file, standard

output to any file or device.

relative In graphics, x and y values that identify the coordinate

location of a pixel by specifying displacements

from some other pixel.

See "comment." **REM**

A word that is used in GWBASIC for a special reserved word

purpose, like a statement keyword, or a function name, etc. It cannot be used as a variable name.

A horizontal line of pixels on the screen. raster

scan code	A number (usually in hexadecimal form) that identifies the position of a key on the keyboard.
scroll	To move all or part of the text display vertically or horizontally so as to show characters that do not fit on the screen.
segment	A 64K-byte area of memory.
sequential access	An access mode in which records are processed in consecutive order, i.e., they are read in the same order in which they were written.
single precision	If a number is not an integer and contains 7 or fewer digits it is a single precision number.
soft key	Synonymous with function key.
soft-key display	The display of the soft-key values on the 25th screen line.
stack	An area of memory to temporarily store data so that the last item stored is the first item to be processed.

An instruction to the computer to perform some sequence of operations.

statement

string expression

An expression that returns a string value.

string variable A simple variable or array element whose value is a string.

subroutine

Section of a GWBASIC program which is called by a GOSUB or ON...GOSUB statement. At the end of the execution of a subroutine, control is returned to the first statement following the most recent GOSUB (or

ON...GOSUB) that has been executed.

subscript

A positive integer number that identifies the

position of an element in an array.

text window A rectangular portion of the screen where text is output. It may be defined by a VIEW PRINT

or a WIDTH statement.

trap

A special form of a conditional breakpoint that is activated by an event to be intercepted. It also refers to the action to be taken after the

interception.

type of variable	Indicates whether the variable is a string or a numeric variable and (if numeric) if it is an integer, a single-precision, or a double-precision variable. The type of variable may be set by a DEF (INT, SNG, DBL, or STR) statement, or by a character definition tag at the end of the variable name.
type of expression	The type of expression is the data-type (string, integer, single-precision, or double-precision) of the resulting evaluation of the expression. It depends on the type of its operands.
typewriter keyboard	The central section of the keyboard that is used as a standard typewriter keyboard.
user function	A function that the user must define before it is called (see DEF FN statement).
variable	A named data item whose values may change during program execution.
variable- length record	A record whose length is independent of the length of other records in the file.
vector	A one-dimensional array.

viewport	A	rectangular	portion	of	the	screen	onto	which
----------	---	-------------	---------	----	-----	--------	------	-------

window contents are mapped. A viewport is defined by a VIEW statement to display both

graphics and text.

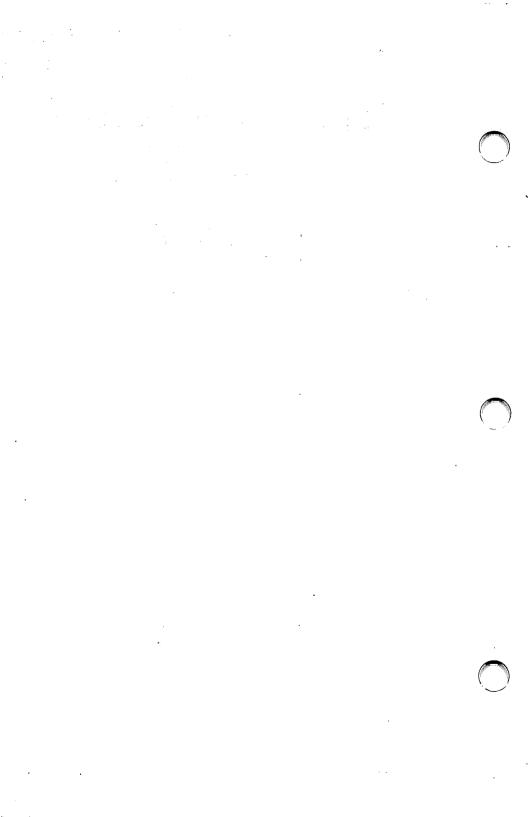
wildcard A special symbol used to represent any single

character (?) or any string of characters (*) in

a filename.

window A rectangular portion of the screen onto which

text may be displayed.



Index

A			
A		CLEAR Command	7-42
ADO D	F 10	CLOSE Statement	7-44
ABS Function	7-16	CLS Statement	7-45
Accessing A Sequentia		COLOR Statement,	
File	4-26	Graphics Mode	7-51
Adding Data to A	4.07	COLOR Statement,	
Sequential File	4-27	Text Mode	7-47
An Exercise in	C 77	COM(n) Statement	7-55
Communication I/O	6-7	Commands For Program	
Arc, Ellipses	5-19	Files	4-18
Arithmetic Operators	3-15	COMMON Statement	7-56
Array Variables	3-9	Communication I/O	6-3
ASC Function	7-17	Communication I/O	
ASCII Codes	A-4	Functions	6-4
ATN Function AUTO Command	7-18 7-19	Concatenation	C-5
	7-19	Constants	3-2
Automatic Line	2-28	CONT Command	7-61
Numbering	2-20	Correcting the Current	0.15
D		Line	2-17
D		COS Function	7-63
BEEP Statement	7-21	Creating A Random	4 00
BINARY to Hexadecin		Access File	4-29
Conversion Table	A-11	Creating A Sequential	4 00
BLOAD Command	7-22	File	4-22
BSAVE Command	7-24	CSNG Function	7-64
		CSRLIN Function	7-65
		Current Directory	4-13
O		CVI,CVS,CVD	7 66
	·26, C-6	Functions	7-66
Calling Subroutine		\boldsymbol{D}	
from GWBASIC	B-6	D	
CALLS Statement	7-27	DATA Statement	7-67
CDBL Function	7-28	DATE\$ Function and	
CHAIN Statement	7-29	Statement	7-69
Character Set	1-9	DEF FN Statement	7-71
CHDIR Command	7-34	DEF SEG Statement	7-73
CHR\$ Function	7-35	DEF USR Statement	7-75
CINT Function	7-36	DEFINT/SNG/DBL/STR	
CIRCLE Statement	7-37	Statements	7-76
		DELETE Command	7-77
		Derived Functions	A-12

DIM Statement	7-78	(i	
Direct Mode	2-3		
Directory Paths	4-9	GET (COM files)	
Displaying Points	5-18	Statement	7-114
Double Precision	3-10	GET (Files) Statement	7-115
DRAW Statement	7-82	GET (Graphics)	
	1-02	Statement	7 - 117
Drawing and Coloring Lines	5-19	GOSUBRETURN	
Lines	9-19	Statements	7-120
E		GOTO Statement	7 - 123
L		Graphics Mode	5-7
EDIT Command	7-86	GWBASIC, Command	7 - 124
END Statement	7-87	GWBASIC Screen	
Entering A Program	2-26	Editor	2-12
ENVIRON Statement	7-88	TT	
		Н	
ENVIRON\$ Function	7-91 7-92		
EOF Function ERASE Statement	7-94	HEX\$ Function	7-132
	1-94	Hexadecimal to Decimal	
ERDEV and ERDEV\$	7.05	Conversion Tables	A-10
Functions	7-95	High Resolution Mode	5-11
ERR and ERL	7.00	How MS-DOS Keeps	
Functions	7-96	Track of Your Files	4-3
Error Messages	D-2	-	
Event Trapping	B-22		
ERROR Statement	7-99	•	
Executing A Program	2-32	IFGOTOELSE	
EXP Function	7-101	Statements	7-133
Expressions and		IFTHENELSE	
Operators	3-14	Statements	7-133
Extended ASCII Codes	A-8	IFTHEN[ELSE]	C-10
		Indirect Mode	2-3
F		Initialization Procedure	2-2
PIELD CL.	7 100	INKEY\$ Function	7 - 137
FIELD Statement	7-102	INP Function	7-139
FILE I/O	C-11	INPUT Statement	7-40
File Numbers	4-5	INPUT# Statement	7-142
FILES Command	7-106	INPUT\$ Function	
FIX Function	7-108	Statement	7-144
FORNEXT	5 100	INSTR Function	7-146
Statements	7-109	INT Function	7-148
FRE Function	7-113	Integer Division	3-17
Function Keys	2-5	Internal Representation	B-4
Functional Operators	3-24		
Functions, Derived	A-12		

Interrupts IOCTL Statement	2-37 7-149	M	
IOCTL\$ Function	7-152	Major Features	1-3
T/		MAT Functions Medium Resolution Mod	C-6 le 5-9
\mathbf{r}		Memory Allocation	B-2
KEY Statement	7-154	Memory Requirements	3-10
KEY(n) Statement	7-160	MERGE Command	7-195
Keyboard	2-4	MID\$ Function and	
KILL Command	7-162	Statement	7-196
\mathbf{L}		MKDIR Command MKI\$,MKS\$,MKD\$	7-200
LCOPY Command	7-163	Functions	7-202
Leaving GWBASIC	2-2	Modes of Operations	2-3
LEFT\$ Function	7-164	Modifying Program Lines	2-20
LEN Function	7-165	Modulus Arithmetic	3-17
Length of Strings	C-3	Multiple Assignments	C-7
LET Statement	7-166	Multiple Statements	C-8
Line Format	1-7	N T	
LINE INPUT		N	
Statement	7-171	NAME	7 000
LINE INPUT#	7 179	NAME Command	7-203
Statement LINE Statement	7-173 7-167	Naming Devices	4-8 4-5
LIST Command	7-175	Naming Files NEW Command	7-205
Listing A Program	2-29	Numeric Constants	3-5
LLIST Command	7-177	Numeric Keypad	2-11
Loading A Program	2-31		
LOAD Command	7-178		
LOC Function	7-179	OCTO P	7 000
LOCATE (Graphics)	7-180	OCT\$ Function	7-206
LOCATE (Text)	5 105	ON COM(n) Statement ON ERROR Statement	7-207 7-209
Statement	7-185	ON ERROR Statement ONGOSUB and	7-209
LOF Function	7-189	ONGOTO	
LOG Function Logical Operators	$7-190 \\ 3-21$	Statements	7-211
LPOS Function	7-191	ON KEY(n) Statement	7-212
LPRINT Statement	7-192	ON PLAY(n)	
LSET and RSET	. 102	Statement	7-215
Statements	7-193	ON STRIG(n)	
		Statement	7-217
		ON TIMER Statement	7-219
		OPEN COM Statement	7-229

OPEN Statement	7-221	Rectangles, Objects,	F 10
Opening		Circles	5-19
Communications Files	6-2	Relational Operators	3-19
OPTION BASE	-	REM Statement	7-279
Statement	7-233	RENUM Command	7-281
OUT Statement	7-234	Reserved Words	1-10
Overflow	3-18	RESET Command	7-283
D		RESTORE Statement	7-284
P		RESUME Statement	7 - 285
DATE OF THE OWNER O		RETURN Statement	7 - 120
PAINT Statement	7-235	See GOSUBRETUR	N
PEEK Function	7-241	RIGHT\$ Function	7-286
PEEKs and POKEs	C-9	RMDIR Command	7-287
PLAY Statement	7-242	RND Function	7-289
PLAY(n) Function	7-246	RUN Command	7-291
PLAY {on off stop}	7-247	Running A Sample	
PMAP Function	7-248	Program	2-34
POINT Function	7-250	•	
POKE Statement	7-252	\mathbf{S}	
POS Function	7-253	B	
PRESET Statement	7-254	SAVE Command	7-293
PRINT Statement	7-255	Saving A Program	2-30
PRINT USING		Scan Codes	A-8
Statement	7-258	Screen Coordinates	5-15
PRINT# and PRINT#	00	SCREEN Function	7-295
USING Statements	7-264	SCREEN Statement	7-297
Protected Files	4-20	Selecting the Screen	. 20.
PSET Statement	7-267	Attributes	5-2
PUT (COM files)	1 201	Sequential Files	4-21
Statement	7-268	SGN Function	7-302
PUT (Files) Statement	7-269	SIN Function	7-303
PUT (Graphics)	1 200	Single Precision	3-5
Statement	7-271	SOUND Statement	7-304
Statement	1-211	SPACE\$ Function	7-304
R		SPC Function	7-308
IV		Special Screen Editor	1-000
Random Access Files	4-28	Keys	2-12
RANDOMIZE	0		7-309
Statement	7-275	SQR Function	1-908
READ Statement	7-277		

STICK Function	7-310	U	
STOP Statement	7-311	II. W G .	
STRIG Statement	1 011	Using Your System	0.00
and Function	7-312	As A Calculator	2-23
STRIG(n) Statement	7-314	USR Function	7-327
STR\$ Function	7-315	USR; calling	B-16
String Dimensioning	C-3	T 7	
String Operators	3-24	V	
STRING\$ Function	7-317	VAL Function	7-329
Substrings	C-4	Variables	3-7
Super Resolution Mode	5-13	VARPTR Function	7-330
SWAP Statement	7-318	VARPTR\$	7-333
Syntax Conventions	1-4		
SYSTEM Command	7-319	VIEW PRINT	3 20,1 000
T		Statement	7-341
T		***	
man n att	7 200	W	
TAB Function	7-320	***	
TAN Function	7-321	WAIT Statement	7-342
Text Mode	5-4	WHILEWEND	
TIME\$ Function and Statement	7-322	Statements	7-343
TIMER Function	7-324	WIDTH Statement	7-345
TIMER Function	1-324	WINDOW	
{ON OFF STOP}		Statement	5-17,7-351
Statements	7-325	WRITE Statement	
TROFF/TRON	1-020	WTIRE# Statement	7-360
Commands	7-326		
Type Conversion	3-11		
Type Conversion Typewriter Keyboard	2-6		