Interactive BASIC Compiler

ZEDCOR

BASIC

Interactive BASIC Compiler

by
Andrew R. Gariepy,
Scott Terry, David Overton,
Greg Branche and Halbert Liang

Documentation by
Michael A. Gariepy

Fourth Edition: 4/87
First edition: 8/85
Second edition: 10/85
Third edition: 5/86

TECHNICAL SUPPORT: 1-(602) 795-3996
Support hours: Monday-Friday, Noon to 5PM, Mountain Standard Time

Zedcor provides free phone support. Be sure to have your invoice number and license agreement number ready. You may need them to get support. Also be sure you understand the problem clearly and describe it is simply as possible. It is usually a good idea to test the problem a few times before calling. Sometimes it's just a syntax problem. Collect or toll free calls will not be accepted for technical support.

In addition, you may contact us on the **GEnie Information Service** by sending electronic mail (EMAIL) to; ZBASIC. We check our mailbox semi-regularly and will respond to you via electronic mail. We also have topics set up on the various Round-Table Bulletin Boards for general information.

Notes on the Fourth Edition

This edition of the ZBasic™ manual contains all the computer appendices. This includes the appendix for MS-DOS™, APPLE™ //e, //c (DOS 3.3 and ProDOS), MACINTOSH™, CP/M™ and TRS-80™ Model 1, 3 and TRS-80 Model 4.

The appendices are at the back of the manual and the new index includes entries for both the reference section and the appendices. It is important to study the appendix for the computer you are using since there are usually enhancements and variations that are important to note.

Acknowledgements

Special thanks to John Kemeny and Thomas Kurtz for creating BASIC, the easiest and most powerful of all the general purpose languages. To Joanne Gariepy for many late hours of editing. An extra special thanks to the programming teams that have meant so much to the success of the product; Scott Terry, Dave Overton, Greg Branche and Hal Liang and to Thomas Dimitri and David Cooper for their help with the MSDOS version. Special thanks to Karen Moesh and Leyla Blisard for making sure ZBasic gets mailed as fast as it does and to Apple Computer, Inc. for the Macintosh™, Laserwriter™, MacDraw, and MacPaint graphic software and to Microsoft for Word™; on which this entire manual was composed and printed (both text and graphics).

Many thanks to the multitudes of ZBasic™ users who provided helpful suggestions for this fourth edition.

Copyright Notice

Trademarks

ZEDCOR, INC.
4500 East Speedway Blvd., Suite 22
Tucson, Arizona 85712-5305
(602) 795-3996
(602) 881-8101
Orders: 800- 482-4567

TABLE OF CONTENTS

TABLE OF CONTENTS

Using ZBasic

Standard Reference

The reference section contains a complete alphabetical list of all Standard ZBasic commands, statements, functions and operators with cross reference to other commands and sections of the manual.

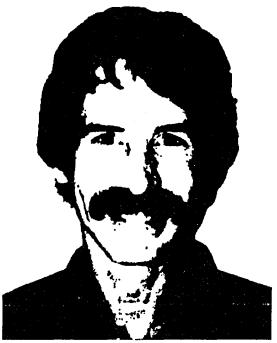
Also see the appropriate appendix for special commands or enhancements for a particular computer model.

TABLE OF CONTENTS

Computer Appendices
VERSION NOTES
Throughout this manual are notes to different versions of ZBasic. An Icon representing the various computer type is used.

Remember the icon for your computer type. If you see the icon in the standard reference manual, a note will follow it describing something of importance for that version.

INTRODUCTION



As the orginal developer of ZBasic and the head of the programming team I want to thank you for your support.

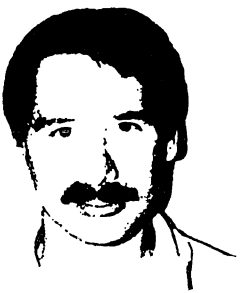
I've been involved in writing ZBasic for eigth years now and am very proud of what we've accomplished. It hasn't been easy but it's sure been fun. How many times does a complex product like ZBasic ever make it to market?

Over the years I have received thousands of suggestions from programmers. I've tried to implement as many of these suggestions as I could. I still need your feedback and comments so I can make ZBasic the most powerful programming tool available. Send your suggestions to the "ZBasic Wish-List Department" or to my attention.

Special thanks to my wife Janis for putting up with my programming late into the night and to the many ZBasic users that have taken the time to send letters of encouragement.

Andrew R. Gariepy
April, 1987

INTRODUCTION



ZBasic has come a long way since it was introduced in 1985. Many thousands of copies, on many different computers, have been distributed all over the planet.

We have accomplished what we set out out to do; to provide a powerful, fast, interactive, simple-to-use, inexpensive BASIC compiler that works the same way on many different computers so you only have to learn a language once.

I've worked hard to make the manual simple to follow and easy to understand.

I highly recommend that you subscribe to the ZBasic newsletter; **"Z"**. It covers all sorts of topics about ZBasic and has listings for public domain ZBasic subroutines on diskette you can get cheap. It's jammed with hints and tricks from other ZBasic users all over the world and from the ZBasic programmers themselves. Call 800-482-4567 to order.

Thank you for your support of ZBasic. Please let us know if you have any ideas of how to improve the product.

Michael A. Gariepy
April, 1987

GETTING STARTED

GETTING STARTED

GETTING STARTED

ZBasic is provided on a diskette for your computer. Before loading ZBasic do the following:

1. Read, sign and return the License agreement in the front of this manual. Keep track of your serial number, you may need it for support.

2. Read the Appendix for your computer. It will explain any variations or enhancements for your version of ZBasic and also has important information about hardware requirements or limitations.

3. MAKE A BACKUP COPY OF THE ORIGINAL ZBasic™ DISKETTE. *Never use the original diskette.* If you do not know how to make backups, refer to your DOS or User Manual.

4. Using the BACKUP, load ZBasic™ according to the instructions for your computer below:

MS-DOS	From A> :	ZBASIC
CP/M-80	From A> :	ZBASIC
TRS-80	From DOS READY:	ZBASIC
Apple DOS 3.3	From FP prompt:	BRUN ZBASIC
Apple ProDOS	From FP prompt:	-/ZBASIC/ZBASIC.SYSTEM
Macintosh	Using the mouse:	*Double Click* ZBasic Icon

HOW TO BE A ZBASIC EXPERT IN TEN MINUTES OR LESS

The following is a quick-and-dirty course that teaches you how to TYPE, RUN, SAVE, QUIT and LOAD a program using ZBasic.

First LOAD ZBasic according to the instructions for your computer above or in your computer appendix. Some versions require that you press <E> to enter the editor. If a prompt appears asking for input, press <E>. See CONFIGURE for more information about the options being offered.

Macintosh users note that the following lessons are done in the COMMAND window.

GETTING STARTED

LESSON ONE: TYPING IN A SIMPLE PROGRAM

When you see the message; **ZBasic Ready,** you may begin entering programs. So we may demonstrate the simplicity of ZBasic, please type in the following program exactly as shown. Always type COMMANDS in UPPERCASE and remember to press <ENTER> or <RETURN> at the end of each line.

```
10 FOR Count = 1 TO 10
20 PRINT "Hi, I'm ZBasic!---"
30 NEXT Count
```

Congratulations, you've just entered your first ZBasic program. To see a listing of the program type: LIST<ENTER>. To find out more about entering and editing programs, see: STANDARD LINE EDITOR . Also see your computer appendix for information about using a full screen editor (if your version has one).

LESSON TWO: RUNNING THE PROGRAM

To run the program you just entered type:

```
RUN
```

The program will print the message; **Hi, I'm ZBasic!---** ten times. ZBasic actually compiles the program but does it so fast that you'll barely notice. When the program is finished you're back in the editor. That's the beauty of interactive compiling.

LESSON THREE: SAVING THE PROGRAM

To SAVE your program, make sure you have an unprotected diskette in the drive and type:

```
SAVE MYPROG
```

The program will be saved to disk for future use.

LESSON FOUR: EXITING ZBASIC

To exit ZBasic type:

```
QUIT
```

You will now be back in the operating system. It's a good idea to save your programs before doing this.

LESSON FIVE: HOW TO LOAD EXISTING PROGRAMS

To load the previously saved program, first re-load ZBasic then type:

```
LOAD MYPROG
```

The program you saved is now back in memory. To see it, type LIST:

```
10 FOR Count = 1 TO 10
20   PRINT "Hi, I'm ZBasic!---"
30 NEXT Count
```

GETTING STARTED

A NOTE TO EXPERIENCED BASIC PROGRAMMERS:

Since the ZBasic Compiler is very similar to the BASIC interpreters found on most microcomputers (except for graphic commands and file I/O), use the Reference Section and your Computer Appendix to check syntax differences from other BASIC's. Use the Index to find more in-depth answers. The appendices in the back of this manual contain the commands and enhancements for specific computers. These appendices are also very useful for converting programs from one machine to another.

If you have been frustrated with incredibly slow interpreters and awkward, complicated compilers, you will be pleased with the power and ease of ZBasic.

A NOTE TO INEXPERIENCED BASIC PROGRAMMERS

This manual is not intended to teach you BASIC programming from scratch. If you lack programming experience we suggest picking up some of the BASIC tutorials for the IBM PC, CP/M systems or the TRS-80, available from most major bookstores and libraries. Once you learn the beginning concepts of BASIC programming, like GOSUB, FOR/NEXT and that type of thing, this manual should be all you need.

ZBasic is very similar to the IBM PC, TRS-80, MSBASIC and GW BASIC interpreters; however, most Graphic commands and Random File commands are different (sequential file commands are very similar).

For those with some experience, this section and the section "Standard Line Editor" are written in a tutorial format.

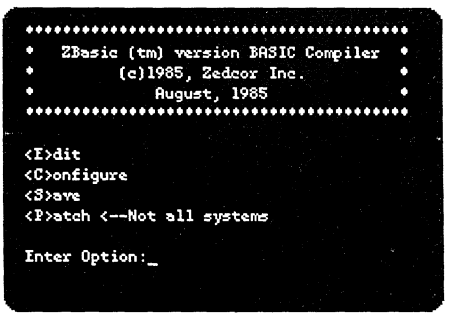
Be sure to examine the appendix in the back of this manual for your computer. It will tell you about any differences and enhancements that are important to know before you start.

CONFIGURATION

CONFIGURATION

CONFIGURATION OPTIONS

Since no two programmers are alike, we allow you to configure your version of ZBasic. Most versions start with a screen something like this:



As you can see below, configuring your version of ZBasic is simple. Simply set the parameters the way you want, then save the reconfigured ZBasic:

`<E>dit` Type "E" to enter the Standard Line Editor . Once in the editor, you may LOAD, TYPE, RUN , EDIT, SAVE or DEBUG your programs.

`<C>onfigure` Typing "C" allows you to configure certain parts of ZBasic. Note that in most cases you will not have to change parameters. See next page for options.

`<S>ave` Typing "S" allows you to save ZBasic with the configuration defaults set to your options. This way you don't have to reconfigure ZBasic every time you load it.

`<P>atch` Typing "P" allows you to make patches to ZBasic. If we make minor changes you won't have to return you disk to us for an upgrade. Not available on all versions.

CONFIGURATION

CHANGING CONFIGURATION

It is simple to change configurations. If the default value is not to your liking simply type in the value you want. Press <ENTER> to skip inputs, Press <BREAK> or <CNTR C> to go back to the main menu.

STANDARD CONFIGURE QUESTIONS		HEX	Decimal	INPUT
1.	Double Precision Accuracy 6-54	ØØØE	ØØØ14	?_
2.	Single Precision Accuracy 2-52	ØØØ6	ØØØØ6	?_
3.	Scientific Precision 2-Double Prec.	ØØ6	ØØØØ6	?_
4.	Maximum File Buffers Open Ø - 99	ØØØ2	ØØØØ2	?_
5.	Array Base Ø or 1	ØØØØ	ØØØØØ	?_
6.	Rounding Number Ø - 99	ØØ31	ØØØ49	?_
7.	Default Variable Type:			
	<S>ingle, <D>ouble, **<I>**nteger		I	?_
8.	Test Array Bounds <Y/N>		N	?_
9.	Convert to uppercase <Y/N>		N	?_
10.	*Optimize expressions as Integer? Y/N		Y	?_
11.	*Space required after Keywords? Y/N		N	?_

* Not all versions.

DEFINITIONS OF THE STANDARD CONFIGURE QUESTIONS

1. Set from six to 54 digits of precision for Double Precision math. Defaults to 14.
2. Set from four up to two digits less than Double Precision. Defaults to 6.
3. Digits of math precision for Scientific functions (ATN, COS etc.)
4. Set the number of files you want OPEN at one time. Up to 99. Two is the default.
5. Array Base Ø or 1. Set zero or one as ARRAY start. Zero is default.
6. Rounding Factor. Sets rounding for PRINT USING and other things.
7. Set variable default to Integer, Single or Double precision.
   Press I, S or D key. Same as DEFDBL, DEFSNG, DEFINT A-Z.
8. Check the runtime program (object code) for array values going out of DIM bounds.
   (Slows the program down but is very good for debugging purposes)
9. Tells ZBasic to convert all lowercase entries to UPPERCASE.
   The variable "FRED" is the same as the variable "Fred" if this is done.
10. Two ways to evaluate expressions. Integer or Floating Point .
    Defaults to integer for speed and size. Set to NO if you want defaults as real.
11. Forcing a space after keywords allows you to embed keywords in variables.

IMPORTANT NOTE: If you change configuration, make sure all CHAINED programs have EXACTLY THE SAME CONFIGURATION. Otherwise unpredictable results may occur.

Macintosh: Select the "Configure" menu item to change or save configuration options. **MSDOS** and **ProDOS** versions of ZBasic have a CONFIG command that allows resetting the options from the Standard line editor. ***CP/M, Apple DOS 3.3** and **TRS-80** versions may not have the last two options offered. Check the appropriate appendix for specifics.

STANDARD LINE EDITOR

STANDARD EDITOR

ZBasic comes with a Standard Editor that works the same way on all computers. While most versions of ZBasic now come with a full screen editor which is easier and faster to use, the Standard Editor allows you to do quick-and-dirty editing and direct commands like an interpreter.

Learning the Standard Editor will allow you to jump from one version of ZBasic to another without having to re-learn the full screen editor for that particular machine.

ENTERING THE EDITOR

Load ZBasic. When the screen says: `ZBasic Ready` you have entered the ZBasic Interactive Programming Environment (a fancy name for the Standard Editor) and may enter programs and type direct commands.

The Standard Line Editor requires each line of a program to have a line number for editing and reference purposes (labels are available too.) Line numbers may range from 0-65534. Each line can be up to 250 characters long. To add a line, type a line number and the text , or use the AUTO command to have ZBasic assign line numbers automatically (some versions of ZBasic will allow you to enter programs without using line numbers. Check your appendix). If you are loading a program without line numbers, they will be added automatically. Line numbers are used for editing in the Standard Line Editor only.

Important Note: Always type keywords and commands in uppercase. Select "Convert to Uppercase" under Configure if you don't want to worry about it.

Important Note: This entire section deals with commands that are to be executed from the Standard Line Editor. If you are in the full screen editor you will need to switch to the Standard Editor. See your computer appendix for Specifics.

This section of the manual refers to the COMMAND window. Switching between the COMMAND and EDIT windows is accomplished with COMMAND E.

STANDARD LINE EDITOR

ENTERING AND DELETING LINES

Type in the following example. Enter it exactly as shown, as we will use this text to illustrate the use of the line editor. Remember to use <ENTER> at the end of each line. This is how ZBasic recognizes a line and stores it in memory:

```
10 THIS IS AN EXAMPLE OF ADDING A LINE
20 THIS IS THE SECOND LINE
30 THIS IS THE THIRD LINE
```

If you make a mistake, use <BACKSP> or <DEL> to delete it. If you <ENTER> a line incorrectly just type it over again. To see the complete program type `LIST`:

LISTING A PROGRAM

To list a line, or range of lines, use **LIST** or just **L**:

YOU TYPE	ZBASIC RESPONDS
`LIST or L`	Lists complete program to the screen
`LIST "SUBROUTINE"`	Lists the line with that label
`LIST "FRED"-`	List all lines after and including the line with the label "FRED"
`LIST 100-200`	Lists lines from 100-200
`LLIST-100`	Lists lines up to 100 to printer
`LIST 100- or L100-`	Lists lines from 100 on
`<period>`	Lists the last line listed or edited
`<UP ARROW>`	Lists previous line (or plus <+> key)
`<DOWN ARROW>`	Lists next line (or minus <-> key)
`L+`	Lists program without line numbers
`LLIST+`	Lists to printer without line numbers
`L+-100`	Lists up to line 100 without showing line numbers
`<SPACE>`	Single steps long listings. <ENTER> continues listing
`</>`	Lists PAGE of lines (10 lines) to screen
`LIST*`	Some systems: Highlights keywords on screen while listing.

DELETING LINES

Deleting lines is accomplished in a number of ways. Examples:

YOU TYPE	ZBASIC RESPONDS
`1000 <ENTER>`	Deletes line 1000
`DEL 1000`	Delete line 1000
`DEL 10-50`	Delete lines 10 through 50
`DELETE 50`	Delete line 50
`DELETE 50-`	Delete line 50 and all lines after
`NEW`	Delete the entire program *Careful!*

NOTE: Labels may be used in place of line numbers (except first example).

ADDING OR INSERTING A NEW PROGRAM LINE

Add or insert a line by typing in a new line number followed by text (be careful not to use the number of a line already being used unless you want to replace it). To insert a line between line 10 and line 20, assign a number such as 15 to the new line (or another number between 10 and 20). To add a line at the end of the program, assign the line a number greater than the largest line in the program.

STANDARD LINE EDITOR

HOW TO EDIT TEXT ON A LINE

The Standard Line editor is used to edit lines in a program and to give commands directly to the compiler. Deleting, inserting, changing or adding new text is easy and fast.

EDIT ANYTHING ON A LINE... EVEN LINE NUMBERS!

Unlike most BASICs, ZBasic allows you to edit anything on a line, even the line number. When a line number is edited, ZBasic creates a new line with that line number. The old line will not be deleted or changed. Very handy for avoiding redundant typing.

The ZBasic line editor functions the same way on all versions of ZBasic. Here are ALL the line edit keys you need to remember:

STANDARD LINE EDITOR KEYS

CURSOR MOVEMENT	DELETE TEXT	INSERT TEXT
<SPACE> Move RIGHT	**<D>**elete one character	**<I>**nsert characters
<BACKSP> Move LEFT	**<K>**ill, Delete upto <letter>	e**<X>**tend line
<S>earch for <letter>	**<H>**ack to end of line	**<Esc>**ape Insert mode
<L>ist the line you are editing	**<Esc>**ape Kill and Hack	

OTHER	
<A>bort changes	**<C>**hange character under the cursor
<ENTER> Keep changes	**<BREAK>** Abort changes (CTRL C on some systems)

CURSOR ARROW keys are often used instead of <SPACE> and <BACKSP>.

MacIntosh: <ESC>=<TAB>, <COMMAND Period>=<BREAK>. **MSDOS and Apple //**: Cursor keys=<SPACE> and <BACKSP>. Delete key also works as <BACKSP>. <CNTRL C>=<BREAK>. **MSDOS**: Insert key = <I>. **CP/M**: <CNTRL C>=<BREAK>. **TRS-80**: <SHIFT up-arrow>=<ESC>.

USING THE LINE EDITOR

The command to edit a line is "EDIT" (or just "E") followed by a line number (or label). If no line number is used, the last line LIST(ed) or EDIT(ed) will be assumed (<COMMA> without <ENTER> will also edit the current line).

"EDIT 20" and "E20" do the same thing.

The following page describes the simple commands used to edit the characters on the line.

STANDARD LINE EDITOR

LEARNING THE COMPLETE STANDARD LINE EDITOR IN 10 MINUTES OR LESS

LISTING THE LINE YOU ARE EDITING <L>

To see the complete line you are editing, and put the cursor at the beginning of the line, press the <L> key. Remember: Line editor commands do not require <ENTER>.

MOVING THE CURSOR ON THE LINE n <SPACE> <BACKSPACE>

To move the cursor back and forth on a line, use <SPACE> or <BACKSP> (<DEL> some systems) (don't use <ENTER>). To move the cursor multiple positions, use a number first.

SEARCH FOR CHARACTER n <S>

To move the cursor to a specific character on a line quickly, use the <S> key, (SEARCH), followed by the character to find. To move the cursor from the "T" in "THIS" to the "L" in "EXAMPLE", just type <S> and <L>.

```
00010 THIS IS AN EXAMPLE OF ADDING A LINE
00010 THIS IS AN EXAMP_
```

CHANGE CHARACTER UNDER CURSOR n <C>

To change the character under the cursor, press <C> followed by the new character. To change five characters, press the <5> key first, the <C> key, then the five keys to replace the old characters.

ABORT (UNDO) CHANGES <A>

To undo changes press the <A> key. All changes, additions and deletions will be aborted.

DELETE CHARACTERS n <D>

To delete characters in a line use the <D> key. Pressing <D> will delete the character under the cursor. To delete five characters press <D> 5 times or press the <5> key and the <D> key.

ESCAPE PRESENT MODE <ESC>

To escape from INSERT, SEARCH, CHANGE, EXTEND or KILL modes, press <ESC>.

DELETE UP TO A SPECIFIC CHARACTER n <K>

To delete, or KILL, a range of characters from the cursor to a specified character, use the <K> key.

INSERT CHARACTERS <I>

To insert text in a line, position the cursor where insertion is desired. Press the <I> key. Type in text or <BACKSP> to erase text. Almost any key may be typed except <ESC>, <ENTER> or <BREAK>.

<ESC>ape exits the INSERT mode.

DELETE TO END OF LINE <H>

To delete all the characters from the cursor position to the end of the line, press the <H> key (Hacks off the remainder of the line).

MOVE TO END OF LINE AND ADD <X>

To move the Cursor to the end of the line and enter the INSERT MODE, press the "X" key (For eXtend). <ESC> will return to the regular line editor mode.

EXIT THE LINE EDITOR <ENTER> or <BREAK>

<ENTER> :	Exit the line edit mode and ACCEPT all changes and additions,
<BREAK>:	To exit the line edit mode and IGNORE all changes and additions

* n is a number. If you type 4D, four characters are deleted. n=nth occurrence or n times.

STANDARD LINE EDITOR

USING OTHER EDITORS OR WORD PROCESSORS

Most versions of ZBasic now come with a Full Screen Editor. Check your computer appendix to see if you have one for your version. If you choose, you may also edit ZBasic programs with a word processor or some other editor. You will need to save the ZBasic program in ASCII using the **SAVE* or SAVE+** commands before editing.

In order for ZBasic to load a text file it requires that:

Line lengths must be less than 250 characters
Every line must be followed by a Carriage Return

If the text file does not contain line numbers, ZBasic will assign line numbers to the program starting with one, in increments of one. Use RENUM to renumber a program. ASCII text takes longer to LOAD and SAVE.

RENUMBER PROGRAM LINES

ZBASIC renumbers lines in a program using the RENUM command.
Format:

RENUM [[*NEW LINE NUMBER*][[, *OLD START*,][*INCREMENT*]]]

YOU TYPE	ZBASIC RESPONDS
`RENUM`	Lines start with 10, Increments of 10
`RENUM 100,,5`	Lines start with 100, Increments of 5
`RENUM 100,20,5`	Renumber From line 20, Start with 100, Increments of 5
`RENUM,,100`	Renumbers all lines by 100

THE CALCULATOR (DIRECT MODE)

ZBasic has a built in calculator. Use "?" or "PRINT" in front of a calculation to see the results. You may also convert number bases like HEX, Binary, Octal and Unsigned Integer. (See BASE CONVERSIONS) Examples:

YOU TYPE	ZBASIC RESPONDS
`PRINT 123.2*51.3`	`6320.16`
`?SQR(92.1)`	`9.5968745`
`PRINT 3/2*6`	`6 (Calculated in INTEGER)`
`?3./2*6`	`9 (Calculated in FLOATING POINT)`
`?320/.0001`	`3200000`

NOTE: Unless you have configured ZBasic to default to floating point , integer is assumed. If configured for "Optimize expressions as Integer", use a decimal point in an expression to force the result of a calculation to be floating point (see CONFIGURE).

STANDARD LINE EDITOR

SAVE, LOAD, APPEND and MERGE

ZBASIC uses the LOAD and SAVE commands to load and save programs. Subroutines saved in ASCII without line numbers may be inserted in your program with APPEND. To SAVE in ASCII use "*" . To SAVE in ASCII without line numbers use "+". Examples:

`SAVE MYPROG`	Saves in tokenized format.
`SAVE CHECKERS 2`	Saves tokenized to TRS-80 drive 2.
`SAVE* MYPROG`	Saves MYPROG in ASCII.
`SAVE+ TEST`	Saves TEST without line#'s in ASCII.
`LOAD CHECKERS`	Loads Checkers.
`LOAD* CHECKERS`	Loads Checkers but strips REMarks and Spaces.
`MERGE MYPROG`	Merges program MYPROG.
`MERGE* MYPROG`	Merges ASCII program, strips REM's and Spaces.
`APPEND 2000 MYSUB`	Loads non-line# ASCII subroutine, MYSUB, to line 2000.
`APPEND* 50 SORT`	Loads SORT to line 50 in increments of 1, strips all REM's and Spaces from the routine.

NOTE: Only non-line numbered ASCII programs may be APPENDED (SAVE+). Only line numbered programs may be merged (SAVE or SAVE*).

When LOAD(ing) programs without line numbers, ZBasic assumes the end-of-line is terminated with <CR>, <CRLF> or 250 characters, whichever comes first. Lines are assigned line numbers starting with one, in increments of one.

FILE DIRECTORY or CATALOG

To see the names of files on the current storage device type **DIR**. Examples:

MS-DOS (also see PATH and CHDIR)
Apple DOS 3.3 and CP/M:

`DIR`	Lists all the files on the present drive
`DIR B:`	Lists the files on drive B
`DIR A:`	Lists all the files on drive A
`DIR C:`	Lists all the files on drive C
	NOTE: The Apple DOS 3.3 version of ZBasic uses A, B, C... for drive specs instead of D1, D2...

APPLE ProDOS: (also see PATH)

`DIR`	Lists all files in current directory
`DIR FRED`	Lists all files in subdirectory FRED
`DIR FRED/TOM`	Lists all files in subdirectory TOM

TRSDOS:

`DIR Ø`	Lists the files on drive zero
`DIR 2`	Lists the files on drive two
`DIR 1`	Lists the files on drive one

Macintosh: (also see FILES$)

`DIR HD30:Fred`	Lists files in folder called "Fred" on root directory called HD30
`LDIR HD30:Fred`	Lists all files to the printer

Be sure to see your COMPUTER APPENDIX for variations.

STANDARD LINE EDITOR

THE MINI-COMPILER (Direct mode similar to an interpreter)

The Mini-compiler permits compilation of one line programs while in the standard editor. This is very convenient for testing logic or math without having to run the entire program. You are limited to one line but may use a colon ":" to divide a line into multiple statements.

Remember to use ? or PRINT to see the results. Examples:

YOU TYPE	ZBASIC RESPONDS
`PRINT LEFT$("HELLO",2)`	`HE`
`PRINT CHR$(65)`	`A`
`PRINT ASC("A")`	`65`
`FOR X=1 TO 500:? X;:NEXT`	`1 2 3 4 5...500`
`? ABS( TAN(1)* EXP(2)+ LOG(9))`	`13.704997622614`
`: LPRINT "HELLO"`	`Prints "HELLO" to the printer`
`PLOT 0,0 TO 1024, 767`	`Plots a line across the screen`
`? &AB`	`171 (HEX to decimal)`

*Note: A Mini-Compiler line may not start with an "E" or "L" since these are used for abbreviations for EDIT and LIST. To do a command that starts with "E" or "L", use a colon ":" first; **:LPRINT**

THE FIND COMMAND

ZBASIC will FIND variables, quoted strings, labels, line numbers and commands within a program quickly and easily. In most cases simply type FIND followed by the text you want to find. The only two exceptions are:

1. To find quoted strings, use one leading quote; **FIND** "HELLO
   Note 1: First characters in quoted string are significant.
   Note 2: "A" and "a" are considered different characters.

2. Use "#" in front of a line number reference; **FIND** #1000

YOU TYPE	ZBASIC FINDS
`FIND "HELLO`	`01010 A=20:PRINT"HELLO THERE"`
`FIND A$`	`01022 Z=1:A$=B$:PRINTA$+B$`
`or...`	`01333 ABA$="goodbye"`
`FIND 99`	`05122 F=2:X=X+2+F/999`
`FIND #12345 (line number)`	`08000 GOTO 12345`
`FIND 100 (not a line number)`	`02000 X=100`
`FIND X(C)`	`03050 A=1:T=ABS(X(C)/9-293+F)`
`or...`	`03044 ZX(C)=4`
`FIND PRINT`	`00230 A=92:PRINTA`
`FIND "SUB5`	`00345 "SUB500": CLS`
`or...`	`03744 GOSUB "SUB500"`
`FIND OPEN`	`03400 OPEN"R",1,"FILE54",23`
`FIND X=X+2`	`09922 F=2:X=X+2+F/999`
`FIND <ENTER>`	`Finds next occurrence`
`<;> (semi-colon key)`	`Finds next occurrence`

To FIND data in remarks or DATA statements use FIND REM ..., FIND DATA ...

Note: If your version of ZBasic comes with a full screen editor, you may have other FIND or REPLACE options. See your computer appendix for specifics.

STANDARD LINE EDITOR

SETTING CHARACTER WIDTH AND MARGINS FOR PROGRAM LISTINGS

ZBasic has powerful formatting commands for making program listings to the screen or printer easier to read.

WIDTH, WIDTH LPRINT and PAGE

Since screen and printer widths vary depending on the hardware, the user may set the width of listing to either the printer or the screen.

COMMAND	RESULT
`WIDTH=0 THROUGH 255`	Sets Screen width for listings.
`WIDTH LPRINT=0 THROUGH 255`	Sets the printer width for listings.
`PAGE 0-255(1), 0-255(2), 0-255(3)`	Formats LINES PER PAGE for printer. (1) Desired lines printed per page (2) Actual lines per page (3) Top Margin

An example of using these commands for printer listings: To set the top and bottom margins to 3 lines each (to skip perforations) and the printer width to 132, type.

```
WIDTH LPRINT=132: PAGE 60,66,3
```

NOTE: WIDTH, WIDTH LPRINT and PAGE may also be used from within a program. Check the reference section for specifics. (In a program, the PAGE function returns the last line printed. The PAGE statement will send a form feed to the printer. A ZERO value disables all the functions above.

AUTOMATIC LOOP AND STRUCTURE INDENTING

For readability, loops are automatically indented two spaces. When WIDTH is set, lines that wrap around will be aligned for readability as in line 10. Completed loops on the same line will show an asterisk at the beginning of the line as in line 120:

LIST+ (without line numbers)

```
CLS: REM THIS IS A LONG
STATEMENT THAT CONTINUES...
FOR X= 1 TO 10
  DO G=G+1
    GOSUB "Graphics"
  UNTIL G=3
NEXT
"MENU"
CLS
END
"Graphics": X=0
DO X=X+16
  PLOT X, 0 TO X, 767
UNTIL X>1023
*FOR X= 1 TO 100: NEXT
RETURN
```

LIST (with line numbers)

```
00010 CLS: REM THIS IS A LONG
      STATEMENT THAT CONTINUES...
00020 FOR X= 1 TO 10
00025   DO G=G+1
00030       GOSUB "Graphics"
00035    UNTIL G=3
00040 NEXT
00050 "MENU"
00060 CLS
00070 END
00080 "Graphics": X=0
00090 DO X=X+16
00100    PLOT X, 0 TO X, 767
00115 UNTIL X>1023
00120*FOR X= 1 TO 100: NEXT
00125 RETURN
```

Note: **LLIST*+** may also be used to do program listings to the Imagewriter or Laserwriter without linenumbers and with keywords highlighted as above.

RUNNING ZBASIC PROGRAMS

RUNNING ZBASIC PROGRAMS

There are a number of ways to compile your programs with ZBasic. The most commonly used is a simple RUN. This lets you compile and debug interactively. Definitions:

RUN COMPILE PROGRAM IN MEMORY AND EXECUTE

The interactive mode is the easiest and fastest way to write and debug your programs. In many ways it is similar to a BASIC interpreter since you may:

1. **RUN a program to check for errors**
2. ***BREAK out of a running program by pressing <BREAK>.**
3. **Return to ZBasic to re-edit the program.**

Interactive compiling is limited to available memory. If a program gets too large you will have to use one of the methods below. ZBasic will tell you when this is necessary with an "Out of Memory" message.

***Most computers require TRON, TRONS, TRONB or TRONX to enable the <BREAK> key. Otherwise pressing <BREAK> may have no affect.**

RUN ***filename*** COMPILE PROGRAM FROM DISK AND RUN

If a program gets too large for interactive compiling using just RUN, the program text may be saved (not in ASCII), compiled, and executed. This is possible because the text to be compiled is no longer resident and frees up memory for the compiled program.

RUN* COMPILE PROGRAM IN MEMORY AND SAVE TO DISK
RUN* ***filename*** COMPILE FROM DISK AND SAVE TO DISK

Compiles the program from memory (RUN*) or disk (RUN*"filename") and saves it to disk. A few moments later ZBasic will request the filename of the resulting compiled program to be saved (For IBM or CP/M use a .COM suffix. For TRS-80 use a /CMD suffix).

This method frees up the most memory for the final program because the source code and ZBasic are no longer resident in memory. Compiled programs saved to disk are machine language programs and should be executed from the operating system like any other machine language program. See column three of the COMPILE MEMORY CHART.

RUN+ COMPILE PROGRAM IN MEMORY AND SAVE AS CHAIN PROGRAM
RUN+ ***filename*** COMPILE FROM DISK AND SAVE AS CHAIN

See CHAINING PROGRAMS for details.

RUNNING ZBASIC PROGRAMS

DETERMINING MEMORY REQUIRMENTS

MEM returns the available memory. (The table may vary on some versions).

TYPE	MEM:	MEANING
00123	Text	Program text memory used (source code).
49021	Memory	Free memory.
00000	Object	Compiled program size of object code.*
00000	Variable	Memory required for variables.*

*Type **MEM** immediately after compiling to get the correct totals. At other times the results of "Object and Variable" may be invalid.

TYPICAL MEMORY USAGE BY "RUN" TYPE

RUN

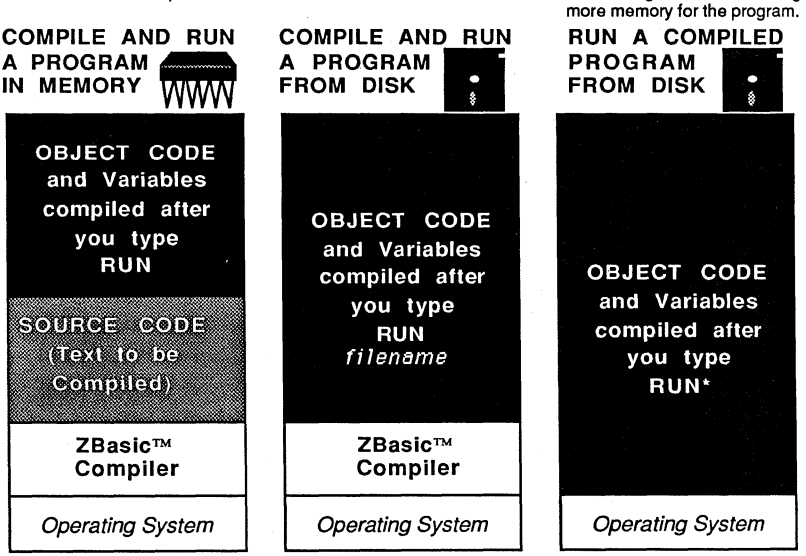
Program text is resident in memory with ZBasic, the compiled program, and the variables used by that program. The user may press <BREAK> when running the program, re-enter the editor and debug any mistakes and re-compile.

RUN *filename*

The program text is saved to disk and compiled from the disk to memory and RUN. Larger programs may be compiled this way because the program to be compiled is not in memory.

RUN* [*filename*]

The program is compiled from memory or disk and the resulting machine language program is saved to disk. The program is executed as a machine language program. When this program is executed the program text and ZBasic are no longer resident, leaving more memory for the program.



*See your Computer Appendix to determine actual memory usage.

RUNNING ZBASIC PROGRAMS

<BREAK>ING OUT OF RUNNING PROGRAMS

To make a program STOP when the <BREAK> key is pressed , use TRON, TRONS, TRONB or TRONX.

TRONB Checks at the start of every line to see if the <BREAK> key has been pressed. If pressed ZBasic returns control to DOS or to the Standard line editor (if in interactive mode). To disable TRONB use the TROFF command.

TRONS Single step trace. CNTR Z to engage/disengage any other key to single step through the program a statement at a time.

TRON Displays line numbers during runtime.

TRONX Checks for the <BREAK> key at the beginning of that line only.

NOTE: TRONX, TRON, TRONS and TRONB may cause INKEY$ to miss keys. TROFF turns all the TRON functions off. All TRONs will slow down programs AND increase size.

USING INKEY$ TO SET BREAK POINTS

You may also use INKEY$ to break out of a program. Put the following line in a program loop or wherever you may want to escape:

IF **INKEY$="S"** THEN STOP
Program will stop if the "S" key is pressed (any key could have been used).

CASES WHERE BREAK WILL NOT FUNCTION

Since ZBasic compiles your programs into machine language, there occurs certain situations where the <BREAK> key will be ignored. Remember; the <BREAK> key is checked only at the beginning of a line. The following example will not break:

```
TRONB
*FOR X= 1 TO 10: X=1: NEXT
```

This is obviously an endless loop (X never gets to 10). One obvious way around this is to avoid putting the entire loop construct on one line.

Examples of other cases where the <BREAK> key is ignored; INPUT, LINE INPUT, DELAY and SOUND statements.

Macintosh: <BREAK> = <COMMAND Period>. <CNTR Z>=< COMMAND <Z>. Most people use BREAK ON instead of TRONB with the Macintosh. See Appendix. **Apple //:** <BREAK> means: <CNTR C>. <CNTR RESET> may be preferable. **MSDOS:** <BREAK> means: <CNTR C>. **CP/M:** <BREAK> means: <CNTR C>: **TRS-80:** <BREAK> means the <BREAK> key.

CHAINING

CHAINING

ZBASIC™

CHAINING PROGRAMS TOGETHER

Chaining is convenient when programs are too large for memory and must be broken into smaller programs. There are three ways to chain programs:

- CHAIN WITH SHARED VARIABLES (GLOBAL or COMMON VARIABLES)
- CHAIN WITH INDEPENDENT VARIABLES
- CHAIN WITH SOME VARIABLES COMMON AND OTHERS NOT

Macintosh CHAIN programs are limited to 28K. See "SEGMENT" and "SEGMENT RETURN" in the appendix for instructions on using the Macintosh memory manager.

EXAMPLES OF CHAINING PROGRAMS WITH SHARED VARIABLES

Programs that will share variables must have those variables defined in exactly the same order in all the programs being chained. ZBasic allows common or shared variables to be DEFINED within **DIM** statements (even if they are not arrays). **CLEAR** or **CLEAR END** should always be used to clear variables that are not shared. Examples:

"STARTB"

```
DIM A(10),100A$(100),Z,F5,W99
OPEN"I",1,"PROG1"          :REM   Always execute this program 1st
RUN 1                      :REM   This is just a starter program
```

"CHAIN1"

```
REM  THIS IS PROG1
TRONB: REM ENABLE <BREAK> KEY
DIM A(10),100A$(100),Z,F5,W99
CLEAR  END
TV=23: PR=4
CLS:  PRINT"THIS IS PROGRAM #1"
PRINT"Z=";Z,"F5=";F5
Z=RND(10):F5=RND(10)
PRINT"Z=";Z;"  F5=";F5
PRINT"JUMPING TO PROGRAM#2"
DELAY 2000
OPEN"I",1,"PROG2"
RUN 1: REM  RUNs Prog2
```

"CHAIN2"

```
REM   THIS IS PROG2
TRONB
DIM A(10),100A$(100),Z,F5,W99
CLEAR  END
ZZ=99: MYVAR=9191
PRINT "THIS IS PROGRAM #2"
PRINT"Z=";Z,"F5=";F5
Z=RND(10):F5=RND(10)
PRINT"Z=";Z;"  F5=";F5
PRINT"JUMPING TO PROGRAM #1"
DELAY 2000
OPEN"I",1,"PROG1"
RUN 1:REM  RUNs Prog1
```

CHAINING

COMPILING THE EXAMPLE PROGRAMS

1. **RUN* STARTB** and save as **START**
   Always **RUN*** a START program. This is a dummy program and is used only to get the chained programs started and contains the runtime routines. Any filename will do.

2. **RUN+ CHAIN1** and save as **PROG1**
3. **RUN+ CHAIN2** and save as **PROG2**

NOTE: Always compile a START program using the RUN* command so that the chained programs have a runtime package. All chained programs must be compiled using RUN+.

USE "DIM" TO DEFINE SHARED OR COMMON VARIABLES

When chained together, both PROG1 and PROG2 will share variables defined on line 10 after the DIM . If F5 equals10 in PROG1, it will still equal 10 when you RUN PROG2.

Because variables "TV" and "PR" are unique to PROG1 and the variables "ZZ" and "MYVAR" are unique to PROG2, **CLEAR END** must be used to initialize them (they must be assigned values). Otherwise false values will be passed from other CHAIN programs.

The example programs (PROG1 and PROG2) will chain back and forth until you press <BREAK>. Lines 80 and 90 are where the programs branch off to the other program.

CLEARING NON-SHARED VARIABLES WHEN CHAINING

Always use **CLEAR END** to clear variables that are not common between the programs. All variables that follow a CLEAR END will be unique to that program and will start out as null values.

(1)
```
10 DIM 200A$(100), 65B$(300)
20 CLEAR END
30 DIM FR(900)
```

(2)
```
10 DIM 200A$(100), 65B$(300)
20 CLEAR END
30 A9=10: Z=33
```

In the above examples, the array variables A$ and B$ are shared and will contain the same values, while all other variables in the program following the **CLEAR END** statement will be null or zero and unique to that program. FR(n) is unique to program (1) and A9 and Z are unique to program (2).

This statement may be used in non-chained programs as well. It is a handy way to null or zero out selected variables (the variables still exist, they are just set to zero or null).

CHAINING PROGRAMS WITHOUT SHARING VARIABLES

This is done exactly as the same as the previous examples for shared variables, except **CLEAR** is used on the first line of each chained program.

In the example programs CHAIN1 and CHAIN2, add a line:

```
3 CLEAR
```

Variables are not shared and **CLEAR** clears all variables (sets them to zero or null) each time a program is entered or chained.

To selectively share some variables and not others use the **CLEAR END** statement described on the previous page and in the reference section.

ERRORS

There are different types of error messages. When errors are encountered during compilation, compiling is stopped and the offending line is displayed. This is a Compile Time error. Errors encountered during execution of a program are called Runtime Errors.

COMPILE TIME ERRORS

After typing RUN, ZBASIC compiles the program. If errors are encountered, ZBASIC will stop compiling and display the error on the screen along with the offending line (when compiling from disk using RUN "Filename" or RUN*, ZBasic will stop compiling, load the Source Code, and LIST the line where the error occurred.) The Statement within the line and the line number will be displayed. The following program would cause ZBASIC to print an error during compile:

```
00010 CLS
00020 PRINT "HELLO THERE MR. COMPUTER USER!"
00030 PRINT "I AM A COMPUTER"
00040 Z=Z+1: X=X+Z: PWINTX

RUN

Syntax Error in Stmt 03 at Line 00040
00040 Z=Z+1: X=X=Z: PWINT X
```

NOTE: The error will be marked in some way depending on the computer system being used. The error marker indicates the general error location on the line where compilation stopped. To edit line 40 above type; EDIT 40 (or just comma). Fix the spelling of PRINT.

ZBasic will often display the missing character it expected.

```
00010 INPUT"Enter a number" A$
RUN
";" expected error in Stmt 01 at line 00010
00010 INPUT"Enter a number"_A$


00010 DIM A(10,10)
00020 A(X)=100
RUN
"," expected error in Stmt 01 at line 00020
00020 A(X_)
```

ERRORS

COMPILE TIME ERROR MESSAGES

A compile time error is one that ZBasic encounters after you type RUN (while it is compiling your program). More often than not, the error is a syntax error. Edit the line to fix the error and type RUN again until all the errors have been deleted.

COMPILE TIME ERROR MESSAGE	DEFINITIONS and POSSIBLE REMEDIES
DIM Error In Stmt...	Only constants may be used in DIM statements: DIM A (X) or Z (A+4) are not allowed. If you have a need to erase and reuse dynamic string arrays see; INDEX$, CLEAR INDEX$, MEM.
No DIM Error In	Array variable being used was not Dimmed. Make sure variable is Dimmed correctly. Most interpreters allow ten elements of an array before a DIM is required. A compiler requires a DIM for every array.
Overflow Error In...	DEF LEN or DIM string length is less than one or greater than 255. Also if CLEAR =zero or CLEAR is too large. Check and adjust range.
Syntax Error In	Anything ZBasic does not understand. Check for spelling, formatting errors and syntax. The offending part of the line is often highlighted.
Too Complex Error...	String function is too complex to compile. Break up complex strings.
Re-DEF Error...	An FN or LONG FN was defined twice.
Variable Error In...	String assignment problem: A$=123: Change to A$=STR$(123)
Out of Memory Error In...	Program is getting too large. Check large DIM statements and defined string lengths, or compile using RUN*. For very large programs you may wish to CHAIN programs together.
Line # Error In...	GOTO, GOSUB, ON GOTO, ON GOSUB, THEN or some other branching command can't find line number or a label.
Mismatch error In...	The assignment to a variable is the wrong type.
Structure Error In ...	FOR without NEXT, DO without UNTIL, WHILE without WEND, LONG IF without END IF or LONG FN without an END FN.
Structure Error In 65535*	Missing NEXT, WEND, END IF, END FN or UNTIL. If unable to find error quickly, LLIST the program. Structures are indented two spaces. Backtrack from the end of the program until the extra indentation is located.
"?" Expected error In ...	ZBasic expected some form of punctuation that was not provided. Check cursor position in displayed line for error.

***NOTE:** Each ZBasic loop command must have one, and only one, matching partner. Each FOR needs a NEXT, each WHILE needs a WEND, each LONG FN needs an END FN, each LONG IF needs an END IF and each DO needs an UNTIL.

ERRORS

RUN TIME ERRORS

A Run Time (execution) error is an error that occurs when the compiled program is running (Object Code). The only Run Time error messages produced are:

DISK ERRORS (Unless trapped by the user). See Disk Errors in the FILES section of this manual.

OUT OF MEMORY ERROR when loading a compiled program saved to disk that is too large to execute in memory.

ARRAY BOUNDS ERROR will be shown if the user configures ZBasic to check for this. This will slow down a program execution but is extremely handy during the debug phase of programming. You may turn this off after the program is completely tested. If access to an array element out of bounds is made , the program is stopped and the line number with the error is printed.

STRING LENGTH ERROR. Some versions of ZBasic have a configure option that tells ZBasic to check for string assignments greater than the length allowed. This does slow execution speed and add memory overhead, so you may want to remove this error checking after the program is debugged. See your appendix for specifics. If an attempt is made to assign a string a value longer than its length , the program is stopped and the line number with the error is printed.

RECOVERING FROM FATAL RUNTIME ERRORS

Since ZBasic is a compiler and converts your code into machine language, there is always a risk that you may unintentionally enter an endless loop or hang up the system (the computer will not respond to anything) .

In these instances you may not be able to get a response from the computer or be able to <BREAK> out of the program. The system may have to be reset or turned off, and back on again to regain control. To avoid losing valuable time, it is very important that you SAVE PROGRAMS and MAKE BACKUPS FREQUENTLY. See your computer appendix for possible alternatives.

USING SINGLE STEP DEBUGGING TO FIND THE SOURCE OF "CRASHES"

Should you encounter a situation where your program goes so far and then the system hangs-up or you get a system error of some kind that you just can't locate, there is a simple way to find the problem.

First put a TRONS and TRON in the program somewhere before the crash occurs. The TRON is added so that you can see a listing of the line numbers as the program executes. Press the space bar a statement at a time, keeping track of the line numbers as they go by.

When the system crashes, make a note of the line number where the crash occurred and fix the problem in your program.

TERMS AND DEFINITIONS

TERMS AND DEFINITIONS

ZBASIC™

TERMS and DEFINITIONS

I use terms throughout this manual that may be unknown to you. The following terms are used to make reading the technical information easier.

IMPORTANT NOTE

"The Hand" is pointing out something of importance for that section. Read it!

OPTIONAL

Items [*enclosed in brackets*] are OPTIONAL. You may or may not include that part of a command, function or statement.

REPETITION

Three periods (ellipsis) mean repetition ... when they appear after the second occurrence of something.

PUNCTUATION

Any punctuation such as commas, periods, colons and semi-colons included in definitions, other than brackets or periods described above, must be included as shown. Any text in Courier font, like this: `COURIER FONT TEXT`, means it is something for you to type in or a simulation of the way it will look on your screen like a program listing.

COMPUTER APPENDIX

Refers to the appendix in the back of this manual, *ABOUT YOUR COMPUTER*.

SPECIAL[32]

The superscripted 32 means this command, function or statement only works on 32 bit computers. See your COMPUTER APPENDIX to see if your computer supports 32 bits. In this edition of the manual it refers to the Macintosh computer only.

ABBREVIATIONS

Frequently used line editor commands have convenient abbreviations:

USE WITH <ENTER>		USE WITHOUT <ENTER>	
?	**PRINT**	, comma	**EDIT** present line
DEL	**DELETE**	. period	**LIST** present line
E	**EDIT**	/ slash	**LIST** next 10 lines
L	**LIST**	; (semi-colon)	**FIND** next occurrence

TERMS AND DEFINITIONS

DIFFERENT (KEY) STROKES FOR DIFFERENT FOLKS

Since ZBASIC operates on many different computers, reference is made to the same keys throughout this manual.

MANUAL USES	YOUR COMPUTER MAY USE
<SPACE>	SPACE BAR
<BACKSP>	BACKSPACE, DELETE, LEFT ARROW
<BREAK>	CONTROL C, COMMAND PERIOD
<ENTER>	RETURN, CARRIAGE RETURN
<ESC>	ESCAPE, CNTRL UP ARROW, TAB
<UP ARROW>	CURSOR UP, PLUS KEY<+>
<DOWN ARROW>	CURSOR DOWN, MINUS KEY<->
<letter>	Press the Key with that letter

See your COMPUTER APPENDIX for variations or enhancements.

LABELS ON LINES

A line may have a label directly following the line number consisting of upper or lowercase, alphanumeric characters, or symbols in any order enclosed in quotes. The length of a label is limited to the length of a line. ZBasic recognizes only the first occurence of a label.

Line numbers are essential only for line EDIT(ing), MERGE, and APPEND. Statements like; LIST, EDIT, APPEND, GOTO, ON GOTO, GOSUB, ON GOSUB, DEL, etc., may use either Labels or line numbers. List programs without line numbers by using LIST+ .

SIMPLE STRINGS

Quoted strings: "Hello", "This is within quotes"

Any String variables: A$, NAME$, FF$, BF$(23).

Any of the following string functions:
MKI$, MKB$, CHR$, HEX$, OCT$, BIN$, UNS$, STR$, ERRMSG$, INKEY$, INDEX$(9).

COMPLEX STRINGS

Complex strings are any combination of SIMPLE STRINGS. Any string operations containing one of the following commands: simple string + simple string, LEFT$, RIGHT$, MID$, STRING$, SPACE$, UCASE$

ZBasic allows only one level of COMPLEX STRING expression. Complex strings MAY NOT be used with IF THEN statements. Convert all multi-level complex strings to simple strings:

CHANGE COMPLEX STRINGS	TO SIMPLE STRINGS
`B$=RIGHT$(A$+C$,2)`	`B$=A$+C$: B$=RIGHT$(B$,2)`
`B$=UCASE$(LEFT$(A$,3))`	`B$=LEFT$(A$,3): B$=UCASE$(B$)`
`IF LEFT$(B$,2)="IT"THEN 99`	`D$=LEFT$(B$,2): IFD$="IT"THEN 99`

The Macintosh version allows much deeper levels of complex strings.

TERMS AND DEFINITIONS

VARIABLE TYPES

A$, A#, A!, A%, and A%(n,n) represent different variables. If no type is given, integer is assumed (unless configured differently by the user or changed with `DEF DBL, DEF SNG or DEF STR`). A and A% would be the same variable. Types:

%	Integer variable
&	4 byte Integer (32 bit machines only)
!	Single precision variable
#	Double precision variable
$	String variable

EXPRESSIONS

Throughout this manual reference is made to *expressions*. There are different types of expressions and the following words will be used to refer to specific expressions.

DEFINITION OF EXPRESSION

EXPRESSION refers to a combination of constants, variables, relational operators or math operators in either integer, floating point or string used to yield a numeric result. The following UNDERLINED examples are EXPRESSIONS.

```
CLEAR 2000

A= T+1

TEST= X^ 2.23* 5+1

IF X*3.4 <= Y*98.3 THEN   Z= 45*84^R

IF A$>B$  AND  B$<>C$  THEN GOTO 1000
```

TERMS AND DEFINITIONS

BYTE EXPRESSION

A **BYTE EXPRESSION** always results in a number from 0 to 255. The expression may be floating point, integer or string, but if the actual result is more than 255 or less than 0, the final result will return the positive one byte remainder. ZBasic will not return an error if the calculation result is out of this range.

INTEGER EXPRESSION

An **INTEGER EXPRESSION** results in an integer number from -32768 to 32767 . The expression may be floating point, integer or string, but if the actual result is more than 32767 or less than -32768, the final result will return the integer remainder which is incorrect. ZBasic will not return an error if the calculation result is out of integer range.

Note: 32 bit computers have a LongInteger range of ± 2,147,483,647.

UNSIGNED INTEGER EXPRESSION

An **UNSIGNED INTEGER EXPRESSION** always results in an unsigned integer number from 0 to 65535. The expression may be floating point or integer but if the actual result is more than 65535 or less than 0 the final result will return the remainder which will be incorrect. See UNS$ for displaying signed integers as unsigned.

Note: 32 bit computers have an unsigned LongInteger range of 0 to 4,294,967,300.

CONDITIONAL EXPRESSION

Conditional expressions like **A=B, A>B, A<B** etc., will return negative one if TRUE (-1), and zero (Ø) if FALSE.

It should be noted that a condition like **IF X THEN...** would be TRUE if X is non-zero and FALSE if X= zero .

IMPORTANT NOTE ABOUT MATH EXPRESSIONS: If you have configured numeric expressions to be optimized as integer, the final result of an expression will be evaluated by ZBasic as integer **UNLESS** one of the following conditions is found within that expression:

* Constant with a type of (#, !, or exponent: D or E)
* Constant with a decimal point (period). Example: .34 or 1.92
* Non-integer variable. (Single or Double precision #, !)
* MATH Functions: COS, SIN, ATN, SQR, LOG, EXP, TAN, VAL, CVB, FRAC, and FIX.
* Floating point math symbols \, ^ or [

Note: One expression may be made up of other expressions within parentheses. Each expression is evaluated separately and must meet the criteria above.

MATH OPERATORS

+	ADDITION
-	SUBTRACTION
*	MULTIPLY
/	DIVIDE
\	DIVIDE (Floating point Divide or Integer Divide)*
	* If configured as "Optimize Expressions as Integer" the \ is forced floating point divide, otherwise it is forced integer divide.
^ or [	EXPONENTIATION (raise to the power)
MOD	REMAINDER OF INTEGER DIVIDE (MODulo)
<<	SHIFT LEFT (BASE^2 MULTIPLY)
>>	SHIFT RIGHT (BASE^2 DIVIDE)

NEGATION

Negation will reverse the sign of an expression, variable or constant. Examples: -A, -12, -.32, -(X*B+3^7), -ABS(Z*R)

SHIFT (binary multiply and divide)

Since computers do internal calculations in binary (BASE 2), SHIFT is used to take advantage of this computer strength. Multiply or divide SHIFTS are faster than floating point multiply or divide and may be used when speed is a factor. (Integer Shift Right loses sign). A good example; `ATN(1)<<2` = Π (instead of the slower; `ATN(1)*4`)

>>n	**Shift right (Divide by 2^n)**
<<n	**Shift left (Multiply by 2^n)**
	(n Truncates to an integer number)

SHIFT FUNCTIONS	BASE 2 Equivalent*	DECIMAL Equivalent	RESULT
4>>1 (Divide)	$4 / 2^1$	4/2	2
4<<1 (Multiply)	$4 * 2^1$	4*2	8
89.34<<2	$89.34 * 2^2$	89.34*4	357.36
.008>>1	$.008 / 2^1$	.008/2	4E-3
999.>>7	$999 / 2^7$	999/128	7.8046875

*2^1=2, 2^3 is the same as 2*2*2, 2^7 is the same as 2*2*2*2*2*2*2
With 10>>8.231 or 10<<8.231 the 8.231 would be converted to integer 8

MATH

REGULAR MATH EXPRESSIONS AND ZBASIC EQUIVALENTS

Regular math and algebraic expressions are quite similar to ZBasic expressions. The user should, however, be aware of some important differences. As in regular algebraic expressions, parentheses determine the part of the expression that is to be completed first. Examples:

Regular Math	ZBasic™ Equivalent
A-2B+1	A-2*B+1
$A(\frac{C}{B})$	A*(C/B)
$(A-B)+T^2$	A-B+T^2
$(AC)^{H^2}$	(A*C)^(H^2)
$(A+\frac{B^2}{C})T^6$	(A+B^2/C)*T^6
A(-B)	A*-B

FORCING EXPRESSION EVALUATION TO DEFAULT TO FLOATING POINT

ZBasic normally optimizes expression evaluation by assuming integer if no floating point types are seen in the expression. This can cause confusion for those used to MSBASIC or other languages without this capability. Setting "OPTIMIZE EXPRESSION FOR INTEGER MATH?" to "NO" sets the expression evaluator to interpret expressions as most other computer languages do; that is, all expressions will default to floating point if parentheses or any part of the expressions contain a floating point operator. While this makes it easier to follow the logic in an expression, the speed of execution time will suffer greatly.

It should be noted that a compiler cannot determine if an expression like C%=A%*B% returns a floating point number. If A%=20000 and B%=20000 an overflow will occur.

NOTE: Some versions of ZBasic, most notably versions older than 4.0, will not allow you to configure the expression evaluator. Older versions default to optimized integer math as described below.

WHY OPTIMIZE EXPRESSIONS FOR INTEGER MATH?

ZBasic defaults to a unique way of interpreting math expressions. Under CONFIGURE, you are given the option of setting expression evaluation to optimized integer or regular floating point. The default is INTEGER. This requires some extra thought on the part of the user but forces programs to execute much faster and much more efficiently.

UNDERSTANDING EXPRESSIONS THAT ARE OPTIMIZED FOR INTEGER MATH

Optimized Integer Expressions return the final result of an expression in integer or floating point, depending on how the expression is evaluated.

To optimize program speed and size, *integer is assumed **UNLESS** one of the following is found in an expression: decimal Point, scientific function, \ (floating point divide: SEE NEXT PAGE DEFINITIONS OF DIVIDE SYMBOLS) , #, ! or a constant > 65,535.

The following examples will give you an idea how ZBasic evaluates expressions as Integer or floating point. (B=10)

EXPRESSION	RESULT	EXPRESSION EVALUATED AS
B* .123	1.23	FLOATING POINT (Decimal point force REAL)
B* 23	230	INTEGER
B *23#	230	FLOATING POINT (# forces Double Precision)
B* 32000	-11264	INTEGER (Overflow error)
B* 32000.	320000	FLOATING POINT (Decimal point)
SIN(B)	-.54402111	FLOATING POINT (Scientific Function)
B*Ø+65535	-1	INTEGER (UNS$(-1)=65535)
B*4800	-17536	INTEGER (UNS$(-17536)=48000)

*Note: You may configure ZBasic to assume floating point by setting "Optimize expressions for integer math" to "NO". See "Configure" in the beginning of this manual.

PARENTHESES IN OPTIMIZED INTEGER EXPRESSION EVALUATION

Parentheses are used to force an expression to be evaluated in a certain order. (See ORDER OF PRECEDENCE)

ZBasic evaluates an expression by examining the outermost portions. In the expression: X*(2*(4.03+4))*5, the innermost portion of 4.03+4 is floating point, but since the outermost portions of X* and *5 are integer, the whole expression is returned as an integer. (B=10 in examples)

EXPRESSION	RESULT	EXPRESSION EVALUATED AS
B*(32000+1)	-7670	INTEGER (Out of range error)
B*(32000.+1)+0!	320010	FLOATING POINT (! forces REAL)
B+(.23)+1200	1210	INTEGER
B+(.23)+1200.	1210.23	FLOATING POINT (period forces REAL)
B+(200*(.001^2))	10	INTEGER
B+200*.001^2	10.ØØØ2	FLOATING POINT
B+ATN(2)	11.107149	FLOATING POINT (Scientific Function)

The expression within each level of parentheses is still evaluated according to the precision in that level.

NOTE: Newer versions of ZBasic may be configured to expression evaluation you are more used to. See "OPTIMIZE EXPRESSIONS FOR INTEGER MATH" above.

MATH

INTEGER AND FLOATING POINT DIVIDE SYMBOLS

It should be noted that the Divide symbols / and \ take on different meanings depending on the type of expression evaluation being used:

Optimized for Integer "YES"	**Optimized for Integer "NO"**
/ = Integer Divide	/ =Floating Point Divide
\ =Floating Point divide	\ =Integer Divide

SCIENTIFIC FUNCTIONS

ZBasic offers several scientific and trigonometric math functions for making many calculations easier.

SQR(expression) **SQUARE ROOT** of expression. Returns the number multiplied by itself that equals expression. SQR(9)=3

LOG(expression) Natural **LOGARITHM** of expression (sometimes refered to as LN(n)). Common LOG10 =LOG(n)/LOG(10)

EXP(expression) Natural logarithm base value: e≈2.71828182845904523560287471352662497757 24 **TO THE POWER** of EXPRESSION. Inverse of LOG.

LOG and EXP may speed up calculations dramatically in certain situations. Some comparative equalities using LOG and EXP:

```
X*Y  =  EXP (LOG(X) + LOG(Y))
X/Y  =  EXP (LOG(X) - LOG(Y))
X^Y  =  EXP (LOG(X) * Y)
```

CONFIGURING SCIENTIFIC ACCURACY

Scientific function accuracy may be configured up to 54 digits of accuracy (32 bit machines may be higher). Default accuracy is 6 digits. Scientific accuracy may be configured from two digits of accuracy, up to Double Precision accuracy (not necessarily the same as Single or Double precision).

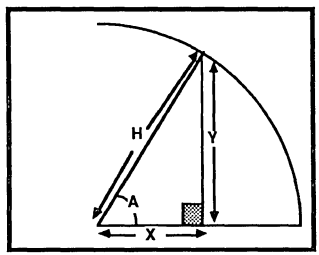
Precision is set when loading ZBasic under <C>onfigure. Scientific math functions are complicated; the more digits of precision used, the longer the processing time required. See "Setting Accuracy" in the floating point section of this manual for information about accuracy, speed charts and memory requirements.

SCIENTIFIC MATH SPEED

When speed is more important than accuracy, configure DIGITS OF PRECISION (under configure at start-up) to 6 digits for DOUBLE, 4 digits for SINGLE and 6 digits for SCIENTIFIC.

MATH

TRIGONOMETRIC FUNCTIONS



TAN(expr) TANGENT of expression in radians.
TAN(A)=Y/X, X=Y/TAN(A), Y=TAN(A)*X

ATN(expr) ARCTANGENT of the expression in radians.
A=ATN(Y/X), Pi=ATN(1)<<2

COS(expr) COSINE of the expression in radians.
COS(A)=X/H, H*COS(A)=X, X/COS(A)=H

SIN(expr) SINE of the expression in radians.
SIN(A)=Y/H, Y=H*SIN(A), H=Y/SIN(A)

SQR(expr) SQUARE ROOT of expression.
H= SQR(X*X+Y*Y)

TAN, ATN, COS AND SIN return results in Radians.

OTHER ZBASIC MATH FUNCTIONS

FRAC(expr) Returns FRACTIONAL portion of an expression
FRAC(23.232)=.232, FRAC(-1.23)=-.23

INT(expr) Returns expression as a whole number
INT(3.5)=3, INT(99231.2)+.0=99231

SGN(expr) Returns the SIGN of an expression
SGN(-23)=-1, SGN(99Ø)=1, SGN(Ø)=Ø

ABS(expr) Returns the ABSOLUTE VALUE of an expression
ABS(-15)=15, ABS(152)=152, ABS(Ø)=Ø

FIX(expr) Returns the whole number of an expression
FIX(99999.23)=99999, FIX(122.6231)=122
(Like INT but forces floating point mode)

expr **MOD** expr Returns the remainder of an integer divide (MODulo)
9 MOD 2=1, 10 MOD 2=Ø, 20 MOD 6=2

RND(expr) Returns a random number between 1 and expr
RND(10) randomly returns:1,2,3,4...10

MAYBE Randomly returns -1 or Ø. (50-50 chance)
IF MAYBE PRINT "HEADS" ELSE PRINT "TAILS"

MATH

DERIVED MATH FUNCTIONS

MATH FUNCTION	TERM	ZBasic EQUIVALENT EQUATION
PI	(ΠI) PI	ATN(1) << 2 (accurate to double precision)
e	e	EXP (1)
Common LOG 10	LOG	LOG(X)/LOG(10)
Area of a CIRCLE	ΠR^2	Y#=(ATN(1)<<2)*Radius*Radius
Area of a SQUARE		Y#=Length*Width
Volume of a RECTANGLE		Y#=Length*Width*Height
Volume of a CUBE		Y#=Length*Length*Length
Volume of a CYLINDER		Y#=(ATN(1)<<2) *Height*Radius*Radius
Volume of a CONE		Y#=(ATN(1)<<2) *Height*Radius*Radius/3
Volume of a SPHERE		Y#=(ATN(1)<<2) *Radius*Radius*Radius*4/3
SECANT	SEC(X)	Y#=1/COS(X)
COSECANT	CSC(X)	Y#=1/SIN(X)
COTANGENT	COT(X)	Y#=1/TAN(X)
Inverse SINE	ARCSIN(X)	Y#=ATN(X/SQR(1-X*X))
Inverse COSINE	ARCCOS(X)	Y#=ATN(1)*2-ATN(X/SQR(1-X*X))
Inverse COSECANT	ARCCSC(X)	Y#=ATN(1/SQR(X*X-1))+(X< Ø)*(ATN(1)<<2)
Inverse COTANGENT	ARCCOT(X)	Y#=ATN(1)*2-ATN(X)
Hyperbolic Sine	SINH(X)	Y#=(EXP(X)-EXP(-X))/2.
Hyperbolic Cosine	COSH(X)	Y#=(EXP(X)+EXP(-X))/2.
Hyperbolic Tangent	TANH(X)	Y#=(EXP(X)-EXP(-X))/(EXP(X)+EXP(-X))
Hyperbolic Secant	SECH(X)	Y#=2./(EXP(X)+EXP(-X))
Hyperbolic Cosecant	CSCH(X)	Y#=2./(EXP(X)-EXP(-X))
Hyperbolic Cotangent	COTH(X)	Y#=(EXP(X)+EXP(-X))/(EXP(X)-EXP(-X))
Inverse Hyperbolic Sine	ARCSINH(X)	Y#=LOG(X+SQR(X*X+1))
Inverse Hyperbolic Cosine	ARCCOSH(X)	Y#=LOG(X+SQR(X*X-1))
Inverse Hyperbolic Tangent	ARCTANH(X)	Y#=LOG((1+X)/(1-X))/2
Inverse Hyperbolic Secant	ARCSECH(X)	Y#=LOG((1+SQR(1-X*X))/X)
Inverse Hyperbolic Cosecant	ARCCSCH(X)	Y#=LOG((1-SGN(X)*SQR(1+X*X))/X)
Inverse Hyperbolic Cotangent	ARCCOTH(X)	Y#=LOG((X+1)/(X-1))/2
Derivative of LN(X) (Natural LOG)		Y#=1/X
Derivative of SIN(X)		Y#=COS(X)
Derivative of TAN(X)		Y#=1+TAN(X)^2
Derivative of COT(X)		Y#=-(1+(1/TAN(X)^2)))
Derivative of ARCSIN(X)		Y#=SQR(1-X*X)
Derivative of ARCCOS(X)		Y#=-SQR(1-X*X)
Derivative of ARCTAN(X)		Y#=1/(1+X*X)
Derivative of ARCCOT(X)		Y#=1/(X*X+1)
Derivative of ARCSEC(X)		Y#=1/(X*SQR(X*X-1))
Derivative of ARCCSC(X)		Y#=-1/(X*SQR(X*X-1))
Derivative of ARCSINH(X)		Y#=1/SQR(1+X*X)
Derivative of ARCCOSH(X)		Y#=-1/SQR(X*X-1)
Derivative of ARCTANH(X)		Y#=1/(1-X*X)
Derivative of ARCCOTH(X)		Y#=-1/(X*X-1)
Derivative of ARCSECH(X)		Y#=-1/(X*SQR(1-X*X))
Derivative of ARCCOSECH(X)		Y#=-1/(X*SQR(1+X*X))

See **DEF FN** and **LONG FN** for adding these math functions to your programs.

ORDER OF PRECEDENCE

In order to determine which part of a math expression is done first an order of precedence is used. The following math operators are performed in the this order.

1. (((1st)2nd)3rd)	Innermost expressions within parentheses always performed first
2. -	Negation (not subtraction)
3. **NOT**	Logical operator
4. ^ or [	Exponential
5. *,/,\,**MOD**	Multiply, Divide, Floating point Divide, MODulo
6. +,-	Addition, Subtraction
7. =,>=,=>,<=,=<, >,<,<>,><	Conditional operators
>>, <<	Shifts
8. **AND, OR, XOR**	Logical operator

ZBasic will calculate each operation of an expression in order of precedence, as defined by the table above. The final result of an expression depends on the order of operations.

If there are items of equal precedence in an expression, ZBasic will perform those operations from left to right.

```
A#=2+5-3*6+1/4.
```

This expression is performed in the following order;

```
1. 3*6
2. 1/4.
3. 2+5
4. (2+5) - (3*6)
5. (2+5-(3*6)) + (1/4.)
```

```
A#=-10.75
```

Important Note: If expressions are optimized for Integer Math, the decimal point after the 4 forces the result of the expression to be floating point. If the decimal point had been omitted, the result would be -11. See CONFIGURE.

MATH

USING PARENTHESES TO FORCE PRECEDENCE

Parentheses are used in math expressions to force ZBasic to calculate that part of an expression first. If a math operation is enclosed in parentheses, which in turn is enclosed within parentheses, the innermost expression will be calculated first.

```
A#=2+5-3*6+1/4.
```

To force the 2+5-3 part of the above equation to be calculated first, and then muliply that by 6 and add 1 second, with division by 4 last, you would express the equation like this:

```
A#=((2+5-3)*6+1)/4.
```

The order of operations in this expression would be:

```
1.  (2+5-3)
2.  (2+5-3)*6+1
3.  ((2+5-3)*6+1)/4.

                              A#=6.25
```

Note: If Expressions are optimized for Integer Math; the outermost expression is used by ZBasic to determine whether the final result will be returned as integer or floating point.

The decimal point after the 4 forces the expression to be calculated as floating point (although each expression within parentheses is evaluated as floating point or integer depending on the rules of expressions). If the decimal point had been omitted the result would have been 6.

To use the standard rules of expression evaluation, set "Optimize Expression evaluation to Integer" to NO under configure. Math expressions will be done in the usual manner if this is done.

CONDITIONAL OPERATORS

The conditional operators return:

0 (zero)	**If the Comparison is FALSE**
-1 (negative one)	**If the Comparison is TRUE**
A non-zero expression	**Is always TRUE**
A zero expression	**Is always FALSE**

These symbols are used for comparing expressions and conditions.

=	**Equal To**
<>,><	**Not Equal To**
<	**Less Than**
>	**Greater Than**
>=, =>	**Greater Than OR Equal To**
<=, =<	**Less Than OR Equal To**

Examples: (A$="HELLO" and A%=2000)

CONDITIONAL EXPRESSION	RESULT
`X=12<20`	`X=-1`
`PRINT 23=45`	`Ø`
`IF 10>5 THEN PRINT "SURE IS"`	`SURE IS`
`IF A%-2000>100-99 PRINT A%`	`Nothing`
`IF VAL(A$)=0 THEN PRINT A$`	`HELLO`
`PRINT 2>5,  3<5,  5>5`	`Ø  -1  Ø`
`IF A%>120 THEN PRINT "OK"`	`OK`
`IF A%*5>=10000 THEN STOP`	`Program STOPs`
`IF A% PRINT "YES"`	`YES (Non zero is True)`
`PRINT 50>50`	`Ø`
`PRINT 50>=50`	`-1`
`IF A%>30000 THEN PRINT "OK"`	`Nothing`
`X=1: IF X THEN PRINT "YEP"`	`YEP`
`X=0: IF X THEN PRINT "YEP"`	`Nothing`
`X=77.321>77.320+1`	`Ø`
`X= "HELLO"="HELLO"`	`X=-1`
`IF A$="HELLO" PRINT "YES"`	`YES`
`IF A$="HELLLO" PRINT "YES"`	`Nothing`
`IF A$>"HEL" THEN PRINT A$`	`HELLO`
`IF A$<>"GOON" THEN PRINT "NO"`	`NO`
`IF STR$(A%)=" 2000" PRINT "YES"`	`YES`

MATH

LOGICAL OPERATORS

Zbasic makes use of the logical operators AND, OR, NOT, SHIFTS and XOR. These operators are used for comparing two 16 bit conditions and binary operations (except on 32 bit computers which can compare 32 bits). When used in comparative operations a negative one (-1) is returned for TRUE, and a zero (Ø) is returned for FALSE.

Logical Operators	RETURNS
condition **AND** condition	TRUE(-1) if both conditions TRUE, else FALSE(Ø)
condition **OR** condition	TRUE(-1) if either or both is TRUE, else FALSE(Ø)
condition **XOR** condition	TRUE(-1) if only one condition is TRUE, else FALSE(Ø)
condition **SHIFT** condition	TRUE(-1) if any non-zero value returned, else FALSE(Ø)
NOT condition	TRUE(-1) if condition FALSE, else FALSE(Ø) if TRUE
EQV (emulate with) NOT (condition XOR condition)	TRUE(-1) if both conditions FALSE or both conditions TRUE, else FALSE(Ø)
IMP (emulate with) (NOT condition) OR condition	FALSE(Ø) if first condition TRUE and second condition FALSE, else TRUE(-1)

AND

```
1 AND 1 = 1
Ø AND 1 = Ø
1 AND Ø = Ø
Ø AND Ø = Ø
```

BOOLEAN "16 BIT" LOGIC

```
      ØØØØØØØ1        ØØØØØ111
AND   ØØØØ1111   AND   ØØØØ1111
=     ØØØØØØØ1   =     ØØØØØ111
```

OR

```
1 OR  1 = 1
Ø OR  1 = 1
1 OR  Ø = 1
Ø OR  Ø = Ø
```

```
      ØØØØØØØ1        1ØØØØ1Ø1
OR    ØØØØ1111   OR    1ØØØØ111
=     ØØØØ1111   =     1ØØØØ111
```

XOR

```
1 XOR 1 = Ø
Ø XOR 1 = 1
1 XOR Ø = 1
Ø XOR Ø = Ø
```

```
      ØØØØØØØ1        1ØØØØ1Ø1
XOR   ØØØØ1111   XOR   1ØØØØ111
=     ØØØØ111Ø   =     ØØØØØØ1Ø
```

SHIFT >>, <<

```
255 >> 2  = 63
23  << 3  = 184
```

```
      11111111        ØØØ1Ø111
>>    ØØØØØØ1Ø   <<    ØØØØØØ11
=     ØØ111111   =     1Ø111ØØØ
```

NOT

```
NOT   1 = Ø
NOT   Ø = 1
```

```
NOT   11ØØ1ØØ   NOT   Ø1111Ø11
=     ØØ11ØØ11   =     1ØØØØ1ØØ
```

With the Macintosh, 32 bit integers may also be used with logical operators (LongInteger&).

NUMERIC CONVERSIONS

INTEGER BASE and SIGN CONVERSIONS

ZBasic has functions for converting integer constants to hexadecimal (BASE 16), octal (BASE 8), binary (BASE 2), unsigned integer and back to decimal (BASE 10). UNS$, HEX$, OCT$ and BIN$ are the functions used to convert an integer to the string representation of that SIGN or BASE.

DECIMAL TO BASE CONVERSION

HEX	OCTAL	BINARY
HEX$(48964)	OCT$(54386)	BIN$(255)
="BF44"	="152162"	="ØØØØØØØ11111111"
HEX$(32)	OCT$(8)	BIN$(512)
="ØØ2Ø"	="ØØØØ1Ø"	="ØØØØØØØ1ØØØØØØØØ"

BASE TO DECIMAL CONVERSION

HEX	OCTAL	BINARY
VAL("&ØØ3Ø")	VAL("&OØØØØ11")	VAL("&XØØØØØØØØØ11ØØØ11")
=48	=9	= 99
VAL("&HFFFF")	VAL("&OØØØØ3Ø")	VAL("&X1111111111111111")
=-1 (65535)	=24	=-1 (65535)

DISPLAYING UNSIGNED INTEGERS

To display or print an unsigned integer number use UNS$. UNS$ returns the unsigned value of the number by not using the leftmost bit as a sign indicator:
UNS$(-1)=65,535, UNS$(-2311)=63,225

ZBasic interprets the integers, -1 and 65,535 as the same value. In BINARY format they are both 1111111111111111. The left-most bit sets the sign of the number to positive or negative. This is the same unsigned integer format used by many other languages.

The same holds true with LongIntegers, only 32 bits are used instead of 16 bits. The signed range is ± 2,147,483,647. The unsigned range is 0 to 4,294,967,293. See DEFSTR LONG in the appendix for ways of using 32 bit HEX$, OCT$, UNS$ and BIN$.

NUMERIC CONVERSIONS

CONVERSION BETWEEN DIFFERENT VARIABLE TYPES

ZBasic will convert variables from one type to another as long as the conversion is within the range of the target variable.

DOUBLE or SINGLE PRECISION VARIABLE =INTEGER VARIABLE will convert exactly (unless single precision is set less then 6 digits).

INTEGER VARIABLE=DOUBLE or SINGLE PRECISION VARIABLE will convert correctly if the double or single precision variables are within the integer range of -32,768 to 32,767 (unsigned Ø to 65,535). Any fractional part of the number will be truncated. Results outside integer range will be the rounded integer result, which is incorrect, and no error will be generated. .

SINGLE PRECISION VARIABLE=DOUBLE PRECISION VARIABLE conversions will be exact to the number of significant digits set for single precision since the calculations are done in double precision. If the single precision default is 6 digits and double precision is 14 digits, the 14 digit number would be rounded down to 6 digits in this example (precision is configurable by the user).

STRING VARIABLE=STR$(INTEGER, DOUBLE OR SINGLE PRECISION VARIABLE) will convert exactly. The first character of the string produced is used for holding the sign. If the number is positive or zero, the first character of the string produced will be a SPACE, otherwise the first character of the string will be a minus (-).

INTEGER VARIABLE=VAL(STRING VARIABLE) will convert correctly, up to the first non-numeric character, if the string variable represents a number in integer range. Fractional portions will be ignored. Zero will be returned if not convertable.

DOUBLE OR SINGLE PRECISION VARIABLE=VAL(STRING VARIABLE) will convert correctly within the range of floating point precision set by the user (rounding will occur if it is more digits than the set precision).

LongINTEGER conversions are the same as regular integers with the exception that the range is much larger. Since all internal integer calculations are done in LongInteger, conversions are simple. See DEFSTR LONG in the Macintosh appendix.

CONSTANTS

CONSTANTS

Constants are values used in expressions, variable assignments, or conditionals. In the following underlined program lines, the constants values remain constant, while values of A$, Z and T are variable.

```
10 PRINT"HELLO THERE":  PRINT A$: Z=Z+T+2322.12
```

ZBasic uses both string (alphanumeric) and numeric constants.

INTEGER CONSTANTS

An integer constant is in the range of -32,768 to 32,767 (or unsigned integer in the range of Ø to 65,535).

The BASE of an integer may be represented in Decimal, Hexadecimal, Octal or Binary. See "Numeric Conversions" for information about converting integers to and from HEX, OCTAL, BINARY and DECIMAL.

MEMORY REQUIRED FOR INTEGER CONSTANTS

Two bytes each in the same format as integer variables.

The Macintosh also has LongInteger constants with a range of ±2,147,483,647. LongInteger constants require four bytes memory each. Macintosh format of integer is the opposite of other versions. i.e. MSB is first and LSB is last.

CONSTANTS

FLOATING POINT CONSTANTS

The range of floating point constants is ±1.0E-64 to ±9.999E+63*. Constants may be expressed in scientific notation and/or up to 54 digits of significant accuracy.

Floating point constants are significant up to the double precision accuracy set by the user. If the number of digits is greater than the accuracy of double precision, it will be rounded to that precision. If the double precision default of 14 digits is assumed, a constant of 1234567890.123456 will be rounded to 1234567890.1235.

Constants may be forced as double or single precision by including a decimal point in the constant or by using # for double precision or ! for single precision.

MEMORY REQUIRED FOR FLOATING POINT CONSTANTS

ZBasic will store floating point constants in Binary Coded Decimal format (See Floating point variables memory requirements). This is based on the actual memory requirement of each constant, with a minimum memory requirement of 3 bytes per constant. To calculate the memory requirements of a specific constant use the formula:

NUMBER of DIGITS in the constant/2+1=Bytes needed*
Minimum of 3 bytes required per Floating point constant.

*The range of Double precision contants is E±16,383 (single precision remains the same for compatibility). To calculate the memory required use the following equation ; Number of Digits/2+2=bytes needed (single precision is the same as above).

Important Note: Some versions of ZBasic offer an optional high speed binary-floating-point option. While the speed of binary math packages is superior, the accuracy, range and memory requirements of binary math are much different from the standard BCD math described above. See the manual provided with the binary math package for details.

STRING CONSTANTS

String constants are alphanumeric information enclosed in double quotes with the number of characters limited by line length (255 characters maximum).

```
"This is a string of characters"
"12345 etc."
"Hello there Fred"
```

Any character except quotes may be included between the quotes. To include quotes in string constants use `CHR$(34)`. `PRINT CHR$(34);"HELLO";CHR$(34)` would print: `"HELLO"`. To conserve memory when using many string constants see **PSTR$**.

MEMORY REQUIRED FOR STRING CONSTANTS

One byte plus the number of characters, including spaces, within the string constant. See PSTR$ for ways of conserving memory with string constants.

VARIABLES

VARIABLES

The word VARIABLE describes the label used to represent alterable values. ZBasic differentiates between four types of variables.

VARIABLE TYPE	TYPE OF STORAGE	RANGE
STRING	ALPHANUMERIC	0 TO 255 CHARACTERS
INTEGER	INTEGER NUMBERS	±32,767
SINGLE PRECISION	FLOATING POINT NUMBERS	E± 63
DOUBLE PRECISION	FLOATING POINT NUMBERS	E± 63

In addition to the variable types described above this version also supports LongInteger and an extended double precision range (single precision is the same as above).

LONG INTEGER	FOUR BYTE INTEGER	± 2 ,147,483,647
DOUBLE PRECISION	FLOATING POINT NUMBERS	E±16,383

Important Note: Some versions of ZBasic offer an optional high speed binary-floating-point option. While the speed of binary math packages is superior, the accuracy, range and memory requirements of binary math are much different from the standard BCD math described above. See the manual provided with the binary math package for details.

VARIABLE TYPE DECLARATION

Variable names may be followed by a type symbol:

$	STRING VARIABLE
%	INTEGER VARIABLE
!	SINGLE PRECISION VARIABLE
#	DOUBLE PRECISION VARIABLE

If type is not given, integer is assumed (unless configured differently). A, A!, A$, A#, A(2,2), A#(2,2), A!(2,2) and A$(2,2) are considered different variables. Note: A and A% are the same variable if ZBasic is configured to Integer.

Type declaration for LongInteger is; &

VARIABLES

DEFINING VARIABLE TYPES

If you want to define variables beginning with a specific letter to be a specific type, use the DEF statement at the beginning of a program.

`DEFSTR A-M,Z`	Defines all variables starting with A thru M and Z as string variables. M and M$ are the same variable.
`DEFSNG A-C`	Defines all variables starting with A thru C as single precision variables. C and C! are the same variable.
`DEFDBL F,W`	Defines all variables starting with F and W as Double precision variables. F and F# are the same.
`DEFINT A,G,T-W`	Defines all variables starting with A,G and T thru W as integer variables. No % needed. A and A% are considered the same variable.

Note: Even if a range of letters is defined as a certain type, a declaration symbol will still force it to be that type. For instance, if A-Z are defined as integer using DEFINT, A$ is still considered a string, and A# is still considered a double precision variable.

`DEFDBL INT A-M`	Defines variables starting with A thru M as LongIntegers. No & needed. A and A& are the same variable.

VARIABLE NAMES

Variable names must have the following characteristics:

- Variable names may be up to 240 characters in length but only the first 15 characters are recognized as a unique variable.
- First character must be in the alpha range of A-Z, or a-z.
- Additional characters are optional and may be alphanumeric or underline.
- Symbols not allowed: ",^/+->=<][()? etc.

SPACE REQUIRED AFTER KEYWORDS

Many versions of ZBasic have this as a configure option. See "Configure". If you don't want to worry about embedding keywords in variables, set "Space Required after Keywords" option to "yes". It will require that keywords be followed by spaces or non-variable symbols. This allows variable names like **FOR**D or **TO**M.

If you do not set this parameter, or do not have this option for your version of ZBasic, you must not embed keywords in variables.

UPPER/LOWERCASE WITH VARIABLES

If you want the variable **TOM** and the variable **tom** to be the same variable, you must configure "Convert to Uppercase" to "yes". See "Configure".

If you do not set this parameter, or do not have this option for your version of ZBasic, you must match case when using variables. i.e. **TOM** and **tom** are different variables.

VARIABLES

MEMORY REQUIRED FOR VARIABLES

VARIABLES	MEMORY REQUIRED
INTEGER %	2 bytes
STRING $	256 bytes (default). String variable length is definable from 1 to 255 characters (plus one for length byte).
SINGLE PRECISION !	4 bytes (default)
DOUBLE PRECISION #	8 bytes (default) If Single or Double precision digits of precision is changed, use this equation to calculate memory requirements: DIGITS of ACCURACY /2+1=BYTES REQUIRED*

ARRAY VARIABLES

ARRAY VARIABLES	MEMORY REQUIRED PER ELEMENT
INTEGER %	2 bytes per element
STRING $	256 bytes (default) per element. String variable length is definable from 1 to 255 characters per element. Add one byte per element to the defined length of the string for the length byte. DEFLEN 200=201 bytes required per element.
INDEX$(element)	1 byte plus the number of characters in an element
SINGLE PRECISION !	4 bytes (default) per element
DOUBLE PRECISION #	8 bytes (default) per element If FLOATING POINT digits of precision are changed, use this equation to calculate memory requirements: NUMBER OF DIGITS/2+1=BYTES REQUIRED*

Note: Remember to count the zero element if BASE zero is used.

Important Note:Some versions of ZBasic offer a high speed binary-floating-point option. While the speed of binary math packages is superior, the accuracy, range and memory requirements of binary math are much different from the standard BCD math described above. See the manual provided with the binary math package for details.

*LongInteger variables and arrays use four bytes each. To determine double precision memory requirements for the Macintosh version: DIGITS/2+2=BYTES REQUIRED per variable or per double precision array element.

VARIABLES

INTEGER VARIABLES

Because ZBasic always attempts to optimize execution size and speed, it will always assume a variable is integer unless the variable is followed by a type declaration (#, !, $, &) or that range of letters has been defined DEFSTR, DEFDBL, DEFDBL INT or DEFSTR. Although it will slow down program performance, you may force ZBasic to assume floating point variables under configuration. See "Configure". **Integer calculations may be 100 to 200 times faster than floating point!**

INTEGER RANGE

-32,768 to +32,767

LongInteger range is ±2,147,483,647. Speed is as fast as regular integers.

DEFINING VARIABLES AS INTEGER

ZBasic assumes all unDEFined variables, or variables without type declarations (#,!,$,&), are integer (unless configured differently by the user).

DEFINT may be used to force a range of variables starting with a certain letter to be integer with the DEFINT statement followed by a list of characters. For example; DEFINT A-G defines all variables starting with A,B, C.....G to be integer. (G and G% would be the same in this case.)

To force a specific variable to be integer, even if that letter type has been DEF(ined) differently, follow a variable with %. TEST%, A% and F9% are integer variables.

INTEGER OVERFLOW RESULTS

If a program calculation in an integer expression exceeds the range of an integer number, ZBasic will return the overflowed integer remainder of that calculation. The result will be incorrect. **ZBasic does not return an Integer Overflow Error.** Check program logic to insure results of an operation remain within integer range.

HOW INTEGER VARIABLES ARE STORED IN MEMORY

Integer variables and integer array elements require two bytes* of memory. To find the address (location in memory) of an integer variable:

ADDRESS^1 = VARPTR(**INTEGER VARIABLE** [(SUBSCRIPT[,SUBSCRIPT[,....])])
ADDRESS^2 = ADDRESS^1 +1

The value of **INTEGER VARIABLE** is calculated using this equation:

INTEGER VARIABLE=VALUE OF ADDRESS^2 *256 + VALUE OF ADDRESS^1

*Requires four bytes for LongInteger. The MSB and LSB are stored in reverse order with regular integers. See the Macintosh appendix for more information.

VARIABLES

FLOATING POINT (BCD) VARIABLES

There are three floating point precisions that may be configured by the programmer to return accuracy up to 54 significant digits:

ZBasic does all BCD calculations in DOUBLE PRECISION. This is extremely important when speed is a factor. If you only need 6 or 7 digits of precision and speed is important be sure to CONFIGURE DIGITS OF ACCURACY AS FOLLOWS:

DOUBLE PRECISION	**= 6**
SINGLE PRECISION	**= 4**
SCIENTIFIC PRECISION	**= 4**

This setting will give you maximum speed in BCD floating point. See the appendix for your computer for variations or enhancements. This is not a factor for the optional binary math package available for some versions of ZBasic.

The Macintosh accuracy can be configured up to 240 digits. Optimum BCD speed is realized by configuring double precision to 8, single and scientific precision to 6.

Important Note: Some versions of ZBasic offer an optional high speed binary-floating-point option. While the speed of binary math packages is superior, the accuracy, range and memory requirements of binary math are much different from the standard BCD math described above. See the manual provided with the binary math package for details.

DEFINING VARIABLES AS SINGLE OR DOUBLE PRECISION

To force the precision of a specific variable to be single precision, follow every occurrence of that variable with an exclamation point (!).

To force a variable to be double precision, follow the variable name with a pound sign (#). To force ZBasic to define a range of variables as double or single precision, use the DEFDBL or DEFSNG statement:

`DEFDBL A-G`	Makes all variables beginning with A-G as Double precision. A# and A would be the same variable in this case
`DEFSNG C`	Makes all variables beginning with C as Single precision. C! and C would be the same variable.

Note: Some verions of BASIC default to single precision variables instead of integer. Use DEFSNG A-Z in programs being converted or configure to assume Floating Point. Also see "Optimize Expression Evaluation as Integer" under "Configure".

VARIABLES

SCIENTIFIC - EXPONENTIAL NOTATION

ZBasic expresses large numbers like:

5Ø,ØØØ,ØØØ,ØØØ
like this: 5E+1Ø or 5E1Ø

The plus sign (+) after the "E" indicates the decimal point moves to the right of the number. Ten places in this example.
Technically: 5*10*10*10*10*10*10*10*10*10*10 or 5*10 ^10.

ZBasic expresses very small numbers like:

.ØØØØØ5
like this: 5E-Ø6

A minus sign after the "E" indicates the decimal point is moved to the left of the number that many places, six in this example. Technically: 5/10/10/10/10/10/10 or 5*10 ^ (-6) .

STANDARD NOTATION	SCIENTIFIC NOTATION
9,123,000,000,000,000	9.123**E+**15 (or E15)
-3,400,002,000,000,000,000	-3.400002**E**18 (or E+18)
.000,000,000,000,000,000,011	1.1**E**-20
-.000,012	-1.2**E**-Ø5

Note: Some BASICs use scientific notation with a "D" instead of an "E". (like 4.23D+12 instead of 4.23E+12) ZBasic will read old format values correctly but will use the more common "E" when printing scientific notation.

WHEN SCIENTIFIC NOTATION IS EXPRESSED

Constants and variables will be expressed in scientific notation when the value is less than .01 or exceeds 10 digits to the left of the decimal point.

You can force ZBasic to print all significant digits in regular notation with: **PRINT USING**

See **PRINT USING** in the Reference Section of this manual.

RANGE OF ZBASIC FLOATING POINT VARIABLES

The range of floating point numbers, *regardless of the accuracy configured* is:

±1E-64 to ±9.9E+63.*

The digits of accuracy are 14 digits for double and 6 digits for single (this is the default for most systems and may be set by the user).

Double Precision exponent may range from E-16,384 to E+16,383. Single Precision exponent is the same for compatibility with 8 and 16 bit machines.

OVERFLOW RESULTS

If an expression results in a number greater then ±9.999E+63, a result of 9.999...E+63 will be returned.

If the number is less then ±1.0E-64 the result will be zero. ***ZBasic will not give an overflow or underflow error.*** Check program logic so that numbers do not exceed floating point range.

BCD FLOATING POINT SPEED

To obtain maximum speed out of BCD floating point math be sure to configure the digits of accuracy to:

DOUBLE PRECISION	**= 6**
SINGLE PRECISION	**= 4**
SCIENTIFIC PRECISION	**= 4**

Normally these setting are fine at 14 and 6 digits. The should only be changed when speed is extremely important. Converting equations to integer will greatly increase speed as well. These settings are important because ZBasic does all calculations in Double precision. Single precision is used for saving memory only.

Important Note: Some versions of ZBasic offer an optional high speed binary-floating-point option. While the speed of binary math packages is superior, the accuracy, range and memory requirements of binary math are much different from the standard BCD math described above. See the manual provided with the binary math package for details.

SINGLE AND DOUBLE PRECISION DIGITS OF ACCURACY

The only difference between Single and Double Precision is that Single Precision holds fewer significant digits than Double Precision. ***ALL ZBASIC FLOATING POINT CALCULATIONS ARE PERFORMED IN DOUBLE PRECISION.***

The default digits of accuracy are 6 digits for Single Precision and 14 digits for Double Precision. The accuracy is configurable from 6 to 54 digits for Double and 2 to 52 digits for Single Precision.*

ACTUAL NUMBER	SINGLE PRECISION*	DOUBLE PRECISION*
12,000,023	12000000	12000023
.009,235,897,4	9.2359E-03	9.2358974E-03
988,888	988,888	988,888
.235,023,912,323,436,129	.235024	.23502391232344
9,999,999 .999,900,001,2	10000000	9999999.9999
88.000,000,912,001,51	88	88.000000912002
12.34147	12.3415	12.34147

*Defaults are 8 and 12 digits for the Macintosh. Both are configurable up to 240 digits.

VARIABLES

ROUNDING

If the digit just to the right of the least significant digit is greater than 5, it will round up, adding one to the least significant digit.

In the example for `.009,235,898,4` above, the last significant 6 digit number is nine, but since the digit after 9 is 7, the 9 is rounded up by one to 10 (and subsequently the 8 is rounded up to 9 to give us 9.2359E-03, which more accurately represents the single precision value. See "Configure" for ways of setting the rounding factor.

NUMBER	DEFAULT ROUNDING FACTOR IS: 49	
####49	.49+.49 = .98 which is less than one	**No Rounding**
####50	.50+.49 = .99 which is less than one	**No Rounding**
####51	.51+.49 = 1 which is equal to one	**Rounds up**
####52	.52+.49 = 1.1 which is greater than one	**Rounds up**

This rounding option will not be available for optional binary floating point packages.

CONFIGURING ACCURACY

ZBasic allows the user to configure the digits of accuracy for single, double or scientific precision functions (like LOG, TAN, SIN, etc).

LIMITATIONS:
Double precision must be at least 2 digits more significant than single. Digits of Accuracy must be in multiples of two (four with Macintosh).

TYPE PRECISION	MINIMUM DIGITS OF ACCURACY	MAXIMUM DIGITS OF ACCURACY*
SINGLE	2 DIGITS	2 DIGITS less than Dbl.
DOUBLE	6 DIGITS	54 DIGITS
SCIENTIFIC	2 DIGITS	54 DIGITS

*Note: **All floating point calculations are done in DOUBLE PRECISION.** For programs where floating point speed is important be sure to set the digits of accuracy to:

DOUBLE PRECISION = 6
SINGLE PRECISION = 4
SCIENTIFIC PRECISION = 4

Important Note: Some versions of ZBasic offer an optional high speed binary-floating-point option. While the speed of binary math packages is superior, the accuracy, range and memory requirements of binary math are much different from the standard BCD math described above. See the manual provided with the binary math package for details.

WARNING: Programs sharing disk files and CHAINED programs with single or double precision variables must have the same accuracy configuration. If one program is set for 6 and 14 digits, and another program is set for 10 and 20 digits, the programs will not be able to read and write each other's files.

Configurable up to 240 digits. For hi-speed set Double to 8, single and scientific to 6.

ACCURACY AND MEMORY REQUIREMENTS

The number of bytes of memory or disk space required for storing single and double precison variables is dependent on the digits of accuracy. If you do not change the accuracy, ZBasic will assume 6 digits for single precision (which requires 4 bytes), and 14 digits for double precision (which requires 8 bytes).*

When you change accuracy, disk files, variables, and constants memory requirements will change as well. The equation to calculate memory or disk file space required for single or double precision variables is:

Digits of Accuracy / 2+1=Bytes required per Floating Point variable

DIGITS of ACCURACY	DISK FILE AND VARIABLE MEMORY REQUIREMENTS
2 digits	2 bytes
4 digits	3 bytes
5 digits	Will round odd digits UP to the next even number, 6 here
6 digits	4 bytes (Single precision default if not configured by user)
.	
.	
.	
14 digits	8 bytes (Double precision default if not configured by user)
.	
.	
.	
52 digits	27 bytes
54 digits	28 bytes

* The Macintosh defaults to 8 digits for single (four bytes) and 12 digits for double (eight bytes). Digits of accuracy are configurable in multiples of four (instead of two as above). To figure memory: Digits of Accuracy / 2+2=bytes required.

WARNING: Different ZBasic programs sharing files and CHAINED programs MUST be set to the same accuracy. Failure to do this will result in program errors, faulty data reads or program crashes.

Important Note: Some versions of ZBasic offer an optional high speed binary-floating-point option. While the speed of binary math packages is superior, the accuracy, range and memory requirements of binary math are much different from the standard BCD math described above. See the manual provided with the binary math package for details.

VARIABLES

HOW BCD FLOATING POINT VARIABLES ARE STORED IN MEMORY

Single precision default is 6 digits (4 bytes). Double precision default is 14 digits (8 bytes). To locate the address (memory location) of either a Single or Double precision variable:

ADDRESS[1]=VARPTR(**FLOATING POINT VARIABLE** [(SUBSCRIPT[,SUBSCRIPT[,..])])

Single and Double precision variables are stored in Binary Coded Decimal format (BCD).

	Bit	7 6 5. . . Ø
*ADDRESS[1]=		• • • • • • • •
	Bit 7:	Mantissa sign (Ø=POSITIVE, 1= NEGATIVE)
	Bit 6:	The exponent sign (Ø=E+, 1=E-)
	Bit 5-Ø:	The exponent value (Ø to 64)
ADDRESS[2]		Digit 1 and 2 (Four bits for each digit)
ADDRESS[3]		Digit 3 and 4
ADDRESS[4]		Digit 5 and 6 (Single precison default)
ADDRESS[5]		Digit 7 and 8
ADDRESS[6]		Digit 9 and 10
ADDRESS[7]		Digit 11 and 12
ADDRESS[8]		Digit 13 and 14 (Double precision default)
.		
.		
ADDRESS[28]		Digit 53 and 54 (Limit of significant digits)

*Single precision defaults to 4 bytes (six digits) and Double precision defaults to 8 bytes (12 digits). Macintosh computers use two bytes for mantissa and exponent for its high precision double precision variable type:

ADDRESS1 & 2	Bit	15 14 13. . . Ø
		• • • • • • • • • • • • • • • •
	Bit 15:	Mantessa sign
	Bit 14:	Exponent sign
	Bit 13-Ø	Exponent value Ø-16383

Range of 32 bit double precision is ±1.0E-16,383 to ±9.999E+16,384.

Note: Single precision range is the same on all machines

Important Note: Some versions of ZBasic offer an optional high speed binary-floating-point option. While the speed of binary math packages is superior, the accuracy, range and memory requirements of binary math are much different from the standard BCD math described above. See the manual provided with the binary math package for details.

ACCURACY VERSUS PROCESSING SPEED

While ZBasic is capable of configuration to extremely high accuracy, you should be aware that calculation time is in direct relation to the number of digits of accuracy.

The following chart will clarify the relationship of processing time to accuracy .

ACCURACY versus PERFORMANCE

Math Function	Relative Speed	Digits of Accuracy 4/6*	6/6*	14	24	36	54	INTEGER
Add/Subract	1		2/3	1	1.20	1.50	2.0	**1/77**
Multiply	3		1/7	1	1.25	3.10	5.8	**1/33**
Divide	12		1/6	1	1.25	1.75	3.0	**1/33**
SQR	50	1/5	1/4	1	2.50	5.75	13.0	
SIN	70	1/5	1/4	1	2.50	5.75	13.0	See USR8(0)
COS	70	1/5	1/4	1	2.50	5.75	13.0	See USR9(0)
TAN	150	1/5	1/4	1	2.50	5.75	13.0	
EXP	100	1/5	1/4	1	2.50	5.75	13.0	
LOG	65	1/5	1/4	1	2.50	5.75	13.0	
ATN	80	1/5	1/4	1	2.50	5.75	13.0	
X ^ n	140	1/5	1/4	1	2.50	5.75	13.0	
X^ (integer)	30		1/2	1	1.67	2.75	5.0	
Shift <<, >>	2		3/4	1	1.25	1.75	2.2	**1/20**

EXPLANATIONS OF HEADINGS

Math Function — The type of math function being timed.

Relative Speed — All speeds are relative to ADD and SUBRACT (SQR takes 50 times longer than add and subtract). The numbers also correspond to the approximate time (in milliseconds) it takes to perform 14 digit math on a Z80 at 4 MHZ.

Digits of accuracy — The numbers under the digits are all relative to 14 digit accuracy. Examples: 54 digit divide takes 3 times longer than 14 digit
6 digit divide takes 1/7th the time of 14 digit multiply.

INTEGER — Integer calculations are relative to 14 digit processing time. Integer add and subtract operations take 1/77th the time of 14 digit operations.

***4/6** — Scientific Accuracy operations were set for LOG, TAN, EXP, ^, SIN, COS and ATN only. Other functions remain at double precision.

SPEED To obtain maximum speed with BCD floating point calculations, configure the digits of precision to: DOUBLE PRECISION=6, SINGLE PRECISION=4, SCIENTIFIC PRECISION=4. ZBasic does ALL calculations in DOUBLE PRECISION .

Important Note: Some versions of ZBasic offer an optional high speed binary-floating-point option. While the speed of binary math packages is superior, the accuracy, range and memory requirements of binary math are much different from the standard BCD math described above. See the manual provided with the binary math package for details.

VARIABLES

STRING VARIABLES

String variables are used for storing alphanumeric, symbol, and control characters.

ZBasic string variables may hold up to a maximum of 255 characters. Any character with an ASCII code in the range of zero to 255 may be used. ASC(A$) will return zero if A$ is a null string: `IF LEN(A$)>0 AND ASC(A$)=0 THEN ASCII CODE=0`

STRING, NUMBER CONVERSIONS

VAL	Converts a string to a number: X=VAL(A$)
STR$	Converts a number to a string: A$=STR$ (43)
CVI, CVB	Converts a condensed string to a number
MKI$, MKB$	Converts numbers to condensed strings

See DEFSTR LONG for using CVI and MKI$ with LongIntegers.

DEFINING STRING VARIABLES

Use a $ symbol following a variable name to make it a string variable. A$ will always be a string variable because of the $.

To define a range of variables beginning with a certain character to be string variables (so you do not have to use $ every time), use the statement DEFSTR:

`DEFSTR A-M`	Makes all variables starting with A, B, C... up to M as string variables. A is the same as A$.
`DEFSTR X,Y,Z`	Makes all variables starting with X,Y and Z as string variables. Z is the same as Z$.

STRING VARIABLE ASSIGNMENTS

String variables are assigned alphanumeric values like this:

```
A$="Hello there"
ART$="VanGogh"+" DaVinci"      (+) connects the strings (concatenates)
Z$=B$
Z$=B$+C$
Z$="Hello"+C$+TEST$
MID$(A$,2,3)="YES"             Puts "YES" into A$ starting at position 2
```

STRING FUNCTIONS AND RELATED COMMANDS

String variables are used for storing and manipulating character information. Here are some examples of ZBasic's string capabilities:

STRING FUNCTIONS	DEFINITION
DIM 10 A$	Sets the string variable A$ to a length of ten.
DEF LEN 20	Sets following strings to 20 character length.
W$=**LEFT$**(A$,3)	W$= 3 characters from the left of A$.
W$=**RIGHT$**(A$,1)	W$= 1 character from the right of A$.
B$=**MID$**(A$,4,2)	B$= 2 characters from A$ beginning at position 4.
**MID$**(A$,2,3)=B$	Puts first 3 characters of B$ into A$ starting at position 2.
C$=**CHR$**(65)	C$= the character represented by ASCII 65 (letter A).
X=**ASC**("A")	X= the ASCII code of "A" (65).
X=**INSTR**(2,A$,B$)	Looks for B$ in A$ starting at position 2, and makes X equal to the position if found, otherwise X= zero.
A$=**STR$**(2345)	Makes A$ equal "2345".
X=**VAL**(A$)	Makes X equal the VALue of A$ (2345 if above).
X=**LEN**(A$)	X= the number of characters in A$.
INPUTA$	Gets input from the keyboard and stores it in A$.
LINEINPUTA$	Accepts any keyboard characters, stores them in A$ and terminates input only with the <ENTER> key.
A$=**INKEY$**	Makes A$= the last key pressed without using <ENTER>.
A$=**UCASE$**("Hello")	Converts A$ to UPPERCASE. (A$ now equals "HELLO").
X=**VARPTR**(A$)	X= the memory address of the variable A$.
WRITE#1,A$;20	Writes 20 characters of A$ out to the disk file#1.
READ#1,A$;20	Reads 20 characters off the disk into A$.
A$=**STRING$**(10,"#")	Makes A$ equal to "##########" .
PRINT **SPACE$**(4)	PRINTs 4 spaces.
SWAP A$,B$	Make A$ equal B$ and B$ equal A$.
LPRINTA$	Prints A$ out to the printer.
PRINT A$	Prints A$ to the screen.
PRINT#2,A$	Prints A$ to disk file 2.
OPEN"R",1,F$,129	Opens the random access file named F$.
KILL A$	Erases the file specified by A$ off the storage device.
A$=**DATE$**	Puts the date into A$ (MM/DD/YY) (Most systems).
A$=**TIME$**	Puts the time into A$ (HH/MM/SS) (Most systems).
A$=B$+C$	Makes A$ equal to B$ plus C$ (Concatenates).
A$="HI"+"THERE"	Makes A$ equal to "HI THERE".
PSTR$	Special command to avoid duplication of string constants.

SPECIAL INDEX$ COMMANDS

INDEX$ (n)="simple string"	INDEX$="Simple string".
INDEX$I (n)=A$	INSERT A$ at INDEX$(n), moves up all other elements .
INDEX$D(n)	DELETE element (n) of INDEX$ and move up other elements.
X=**INDEXF**(A$)	Looks for A$ in INDEX$ (all) X equals element if A$ found. else X equals -1.
X=**INDEXF**("END",950)	Look for "END" in INDEX$ starting at the 950th element.
CLEAR nnnnn	Set aside nnnnn bytes for INDEX$.
CLEAR INDEX$	Nullify the contents of the entire INDEX$ array.

VARIABLES

STRING CONDITIONALS

Strings may be compared using conditional operators just like numbers. The difference is that they are compared by the value of the ASCII code for that number. For instance, the ASCII code for "A" is 65 and "B" is 66. Therefore the expression "A"<"B" would be true (-1).

See ASCII Chart in your computer manual. ASCII characters may vary from computer to computer and from printer to printer.

Be aware that ZBasic differentiates between upper and lowercase characters. "a" is greater than "A" because the ASCII code for "a" is 97 and the ASCII code for "A" is 65. If you want ZBasic to look at a string variable as uppercase only, use the UCASE$ function to convert it.

ZBasic "looks" at all the characters in a string when doing comparisons. "Aa" is greater than "AA". "AAAAAAa" is greater than "AAAAAAA" etc. ZBasic will compare characters in a string to the last character in that string.

CONDITION	RESULT
"RRRRR"<"S"	True (-1)
"FRANK"="FRANK"	True (-1)
"abc">"ABC"	True (-1)
TEST$="Hello"(If TEST$="Hello")	True (-1)
"A">"B"	False (Ø)
"YES"="yes"	False (Ø)

SIMPLE STRINGS

Quoted string: `"Hello", "This is within quotes"`
String variable: `A$, NAME$, FF$, BF$(2,3)`
Any of the following string commands: MKI$, MKB$, CHR$, HEX$, OCT$, BIN$, UNS$, STR$, ERRMSG$, TIME$, DATE$, INKEY$, INDEX$(n)

COMPLEX STRINGS

May be any combination of SIMPLE STRINGS.

String operations containing one of the following commands: simple- string + simplestring, LEFT$, RIGHT$, MID$, STRING$, SPACE$, UCASE$ would be a complex string.

COMPLEX STRINGS MAY NOT BE USED WITH IF-THEN STATEMENTS.

ZBasic allows only one COMPLEX STRING per statement. If you wish to perform more than one complex string at a time, simply divide the complex string expression into multiple statements like this:

CHANGE complex strings

```
B$=RIGHT$(A$+C$,2)
B$=UCASE$(LEFT$(A$,3))
IF LEFT$(B$,2)="IT" THEN 99
```

TO simple strings

```
B$=A$+C$:  B$=RIGHT$(B$,2)
B$=LEFT$(A$,3): B$=UCASE$(B$)
D$=LEFT$(B$,2): IFD$="IT" THEN 99
```

VARIABLES

USING STRING VARIABLES EFFICIENTLY

String variables will require 256 bytes of memory for each string used if the string lengths are not defined by the user. It is important to realize that extensive use of string variables or string array variables may require the user to define string lengths to avoid running out of memory.

Note: Some BASIC(s) have what is referred to as "Garbage collection". ZBasic's method of storing strings NEVER creates time wasting "Garbage Collection".

DEFINING THE LENGTH OF STRING VARIABLES

ZBasic strings have a default length of 255 characters. This can cause excessive memory usage. To obtain maximum memory efficiency, there are two ways of defining the length of string variables and string array variables:

DEF LEN = *number* (Numbers only. *No expressions.*)
DIM *number STRING VARIABLE*, or *number STRING ARRAY*, ...

DEFINING STRING LENGTHS WITH DIM

```
DIM X$(10), 20 A$, Z$(5), 45 TEST$, 10 MD$(20,20)
```

In this example the strings are allocated:

X$(10)	255 each element (255 is the default. 2816 bytes)
A$	20 (21 bytes)
Z$(5)	each element of Z$ as 20* (21*6=105 total bytes of memory used.)
TEST$	45 (46 bytes)
MD$(20, 20)	each element of MD$(20,20) as 10. (21 * 21 *11=4851 total bytes of memory used.)

* If no length is defined, the last given length in that DIM statement is used (20 for A$ in this example). If no length was defined in that DIM statement then the **DEF**ined **LEN**gth is assumed (255 if the string length has not been previously defined)

Note: Add one to the defined length of each string to determine the actual memory requirement of the string *PLUS ONE* for the *LENGTH BYTE*.

VARIABLES

DEFINING STRING LENGTHS WITH DEFLEN

Another command for DEF(ining) the LEN(gth) of string variables is:

DEF LEN = NUMBER *(No expressions)*
(In the range of 1 to 255)

Each string variable located AFTER the statement will have that length, unless another DEFLEN or DIM statement is used.

```
DIM A$(9,9), X(99),  H#(999), 4Bull$
DEF  LEN=50:B$="HOPE"
C$="HELLO"
DEF  LEN=100
ART$="COOL"
DIM Coolness$(9)
A$=ART$
```

In the example:

A$(9,9)	allocated **255** characters for each array element (ZBasic automatically allocates 255 if length has not been defined).
Bull$	allocated 4 characters.
B$** and **C$	allocated 50 characters each.
ART$	allocated 100 characters.
Coolness$	allocated 100 characters for each element.
A$	allocated 100 characters.

Note: The actual memory required for each string (each string element in an array) is the defined length plus one byte for the length byte.

HOW STRING VARIABLES ARE STORED IN MEMORY

ADDRESS=VARPTR(**STRING VARIABLE** [(SUBSCRIPT[,SUBSCRIPT[,....])])

ADDRESS	Length Byte: Holds number of characters in the string.
ADDRESS+1	First character of the string variable
ADDRESS+2	Second character
.	
.	
.	
ADDRESS+**n**	Last character of the string variable
ADDRESS+255	Last address available for undefined string variable
ADDRESS+Defined Length	Last address available for defined string variable

WARNING 1: Strings should never be assigned a character length longer than the assigned length. If the length of A$ is 5 and a program line is executed that has: A$="1234567890", the characters "6" through "0" will overwrite the variables following A$, possibly causing system errors or faulty data.

WARNING 2: If using INPUT to input strings with set length, always make sure the string length is at least one longer than the length being used for input.

For most versions of ZBasic, no error is generated if string assignments exceed the length of the string.

See "Configure" in the Macintosh appendix for setting string length error checking.

INDEX$

INDEX$

SPECIAL INDEX$ STRING ARRAY

INDEX$ is a special ZBasic string array with some powerful and unique capabilities.

The following commands work with INDEX$ variables only.

INDEX$ COMMAND	MEANING
INDEX$(n)=*simple string*	Assigns a value to INDEX$(n)
INDEX$ I(n)=*simple string*	Move element n and all consecutive elements up one and **INSERT***simple string* at element n (the value in element 3 moves up to element 4...). Actually inserts the value into the array without destroying any other elements.
INDEX$ D(n)	**DELETE** element n and move all consecutive elements back down to fill the space (value in element 4 moves down to element 3...).
X=**INDEXF**(*simple string* [,*start#*])	**FIND** simplestring in INDEX$. Begin looking at element START#. **If found X=element number If not found X = -1.**

USING INDEX$

INDEX$ array variables may be assigned values like other string variables. To illustrate the power of INDEX$, the following values have been stored into INDEX$ elements INDEX$(0) through INDEX$(3) and will be used in the examples on the following pages:

ELEMENT #	DATA
INDEX$(0) =	"AL"
INDEX$(1) =	"BOB"
INDEX$(2) =	"DON"
INDEX$(3)=	"ED"

INDEX$

INSERTING ELEMENTS INTO INDEX$

INDEX$ **I** (n) To INSERT "CHRIS" into INDEX$, between "BOB" and "DON", you would use the command INDEX$ I(2)="CHRIS".

This instructs ZBasic to move "DON" and "ED" down and insert "CHRIS" in element 2. (INDEX$ I(2)=A$ would also be legitimate) INDEX$ would now look like this:

ELEMENT #	DATA
INDEX$(0) =	"AL"
INDEX$(1) =	"BOB"
INDEX$(2) =	"CHRIS"
INDEX$(3) =	"DON"
INDEX$(4) =	"ED"

DELETING ELEMENTS FROM INDEX$

INDEX$ **D** (n) To DELETE "BOB" from INDEX$ use the command INDEX$ D(1). This instructs ZBasic to delete element one, and move "CHRIS" and "DON" and all the other elements up to fill in that space. The INDEX$ array would now look like this:

ELEMENT #	DATA
INDEX$(0) =	"AL"
INDEX$(1) =	"CHRIS"
INDEX$(2) =	"DON"
INDEX$(3) =	"ED"

FIND A STRING IN INDEX$

X=**INDEXF**(*simplestring* [,*element n*])

ZBasic will begin searching from element **n** (element zero if not specified) for the string specified by simple string. Examples:

IF FOUND	IF NOT FOUND
X=ELEMENT NUMBER	X=NEGATIVE ONE(-1)

To FIND "DON" in the above list let's say that A$="DON". Using the command X=INDEXF(A$), X would return 2 to show that "DON" is in element 2 of INDEX$.

To FIND "CHR" (part of "**CHR**IS"), you would use the command X=INDEXF("CHR"). X would return with the value of 1 since a match was found in the first three characters of "**CHR**IS".

If you tried to FIND "RIS": X=INDEXF("RIS") , X would return with a value of -1 (negative one) since the FIND command begins the search at the first character of each element, which MUST be significant ("C" must be part of the search).

If the command had been INDEXF("CHRIS" , 3), X would have equaled -1 since the search began at element 3 and "CHRIS" is at element 1 it would never find "CHRIS."

INDEX$ MEMORY REQUIREMENTS

INDEX$ variable elements use memory only if there are characters stored in that element and only as much memory as needed to hold those characters (plus one for length byte). CLEAR nnnnn is used to allocate memory for INDEX$. **CLEAR INDEX$** will clear (nullify) the present contents of INDEX$.

INDEX$ LIMITATIONS

INDEX$ may not be used with SWAP.

USES OF INDEX$

INDEX$ is a valuable tool for disk indices, in-memory data bases, creating word processors, holding lists of strings with varying lengths and much more.

INDEX$ is especially useful anytime unknown string elements lengths are needed.

USING INDEX$ FOR AN INSERTION SORT

A good example of the power of INDEX$ is using it to create a perpetual sort. It allows you to add items to a list instantly and always have the list in order:

```
CLEAR 10000: TRONB
DO
  INPUT"Input String";A$: GOSUB "INSERTION SORT"
UNTIL A$="END" <--- Type END to end inserting
GOTO "PRINT LIST"
:
"INSERTION SORT"
REM N=Number of items
REM A$= New to string to insert
:
B=N: S=0
DO
  H=(B-S+1)>>1.
  LONG IF A$ <= INDEX$(B-H)
    B=B-H
  XELSE
    S=S+H
  END IF
UNTIL B=S
INDEX$ I(B)=A$
N=N+1
RETURN
:
"PRINT LIST"
FOR X=1 TO N
  PRINT INDEX$(X)
NEXT
END
```

INDEX$

HOW INDEX$ ARRAY VARIABLES ARE STORED IN MEMORY

The INDEX$ array is stored in memory in one contiguous block. The distance between each element is the number of characters in the string plus one byte for the length byte of the string.

WARNING: It is suggested that strings in INDEX$ not be manipulated with PEEK and POKE.

Note: CLEAR is used on some computers to allocate memory for INDEX$. CLEAR INDEX$ is used to nullify the contents of INDEX$

This version has the ability to use up to ten INDEX$ arrays at the same time. See appendix for details. Also see MEM(-1) for determining memory remaining for INDEX$.

ARRAY VARIABLES

ARRAY VARIABLES

An Array variable is a multi-celled variable followed by coordinates for specifying which cell is to be used. The following is an example of a one dimension string array with 101 elements.

ARRAY ELEMENT	VALUE
NAME$(Ø)=	"ABE"
NAME$(1)=	"ADAM"
NAME$(2)=	"ALEX"
NAME$(3)=	"AMOS"
.	
.	
.	
NAME$(100)=	"ZORRO"

Separate variables could be used for each value, like NAME1$="ABE", NAME2$="ADAM"... but typing a hundred different variables would become very tiring.

Array variables are much easier to use when inputting, saving, loading, printing long lists, moving data around in a list, sorting lists of information, etc. This example shows how easy it is to print a complete list of the names in the array of variables.

```
FOR X =Ø TO 1ØØ
  PRINT NAME$(X)
NEXT
```

Computers are very good at manipulating large amounts of data and using regular variables to do this is very impractical.

MULTI-DIMENSIONED ARRAYS

ZBasic will allow arrays of 1, 2, 3 or more dimensions, depending on the amount of memory available on your computer.

ARRAY VARIABLES

TWO DIMENSION ARRAY EXAMPLE

The following chart shows a two dimensional integer array; **A(3,3)**. The number of elements are determined by the BASE OPTION that was configured when loading ZBasic. The default is Base 0:

A(3,3) BASE 0 dimensions are 4 elements down (0,1,2 and 3) and 4 elements across (0,1,2 and 3). Base zero utilizes all the elements including the italicized.

A(3,3) BASE 1 dimensions are 3 elements down (1,2,3) and 3 elements across (1,2,3) (not the italicized):

TWO DIMENSION ARRAY			
A(0,0)	*A(1,0)*	*A(2,0)*	*A(3,0)*
A(0,1)	A(1,1)	A(2,1)	A(3,1)
A(0,2)	A(1,2)	A(2,2)	A(3,2)
A(0,3)	A(1,3)	A(2,3)	A(3,3)

This array was DIM(med) A(3,3). A(1,3) represents the cell underlined above. Accessing a cell only requires giving the correct coordinate after the variable name.

Variables, constants or expressions may be used in specifying coordinates:

```
A(3,2),     A(X,Y),     A(2,X),     A(X*2/3,2+Y).
```

BASE OPTION

Zero is considered an element unless you set the BASE OPTION to one when configuring ZBasic. See "Configure" for more information about setting the Base option. The default BASE is zero.

DEFINING THE DIMENSIONS OF AN ARRAY

All variable arrays MUST be **DIM**ensioned at the beginning of a program. When you RUN a program, memory is set aside for the array based on the number of elements you have **DIM**ensioned.

An example of DIM:

```
DIM A%(10,10,10), A#(5), A!(9,7), B$(10), 5Cool$(20)
```

Only numbers may be used within DIM statement parentheses. The following DIM expressions are ***illegal***:

```
DIM A(X),     A(2*X),     A(FR).
```

ARRAY VARIABLES

HOW ARRAYS USE MEMORY

The following chart shows how to calculate the memory requirements of the arrays **DIM**ensioned above with a BASE OPTION of zero (default value).

ARRAY	Type	Bytes per Element	How to Calculate**	Memory Required
A%(10,10,10)	INTEGER	2	11*11*11*2	2662 Bytes
A# (5)	DOUBLE PREC.	8	6*8	48 Bytes
A! (9,7)	SINGLE PREC.	4	10*8*4	320 Bytes
B$(10)	STRING	256	11*256	2816 Bytes
Cool$(20)	STRING	6	21*6	126

**Note: If you use a BASE OPTION of ONE, you will not need to add one to the dimension. For instance, in the first example the way to calculate the memory required would be: 10*10*10*2. Also see DEF LEN and DIM under STRING VARIABLES for info about defining string lengths.

Macintosh also has LongInteger arrays. Each element takes 4 bytes.

ARRAY BOUNDS CHECKING

During the initial stages of writing a program, it is a good idea to configure ZBasic to check array bounds in runtime. See "Configure" for more information.

OUT OF MEMORY ERROR FROM DIMMING

It is necessary to have an understanding of how arrays use memory. DIMensioning an array larger than available memory will cause ZBasic to give an OUT OF MEMORY error at Compile time or RUN time. When calculating large arrays be sure to check if memory is sufficient.

ARRAY VARIABLES

PRINTING ARRAYS

Arrays were designed to make manipulating large lists of data easy. The following routines print the values of ARRAY(50) and/or ARRAY(50,5) to the screen (Substitute LPRINT for PRINT or use ROUTE 128 to print to the printer). Use AUTO or make your own line numbers. It does not matter which numbers are used.

"One Dimension array PRINT routine"

```
DIM ARRAY(50)
FOR X=Ø TO 50
  PRINT ARRAY(X)
NEXT
```

"Two Dimension array PRINT routine"

```
DIM ARRAY(50,5)
FOR X=Ø TO 50
  FOR X2=Ø TO 5
    PRINT ARRAY(X,X2),
  NEXT X2
    PRINT
NEXT X
```

MAKING AN ENTIRE ARRAY ONE VALUE

The following examples show how to make an entire array (ARRAY(50) or ARRAY(50,5)) equal to a certain value. This would be convenient if you wanted to zero out an array or have all the elements start the same values.

"One Dimension array ASSIGNMENT routine"

```
DIM ARRAY(50)
FOR X=Ø TO 50
  ARRAY(X)=VALUE
NEXT
```

"Two Dimension array ASSIGNMENT routine"

```
DIM ARRAY(50,5)
FOR X=Ø TO 50
  FOR X2=Ø TO 5
    ARRAY(X,X2)=VALUE
  NEXT X2
NEXT X
```

ARRAY VARIABLES

USING ARRAYS FOR SORTING

Arrays are also very convenient for organizing large lists of data alphabetically or numerically, in ascending or descending order.

The first program below creates random data to sort. This program is for example purposes only and should not be included in your programs. **These programs are included on your master disk.**

Follow the GOSUB with the label of the sort routine you wish to use (either "QUICK SORT" or "SHELL SORT"). Any line numbers may be used. These sort routines may be copied and saved to disk (using SAVE* or +) as a subroutine to be loaded with APPEND. See APPEND.

SORT.BAS FILL ARRAY WITH RANDOM DATA FOR SORTING

```
DIM SA(500), ST(30,1):   REM ST (30,1) FOR QUICK SORT ONLY.
NI=500:                  REM Change DIM 500 and NI if sort larger
FOR X=0TO NI
  SA(X)=RND(1000):       REM  Stores random numbers for sorting
NEXT
PRINT"Start Time:";TIME$
GOSUB  "QUICK  SORT":    REM Or SHELL SORT
PRINT"Finish Time:";TIME$
FOR X=NI-10 TO NI
  PRINT SA (X):          REM Print last to make sure SORT worked.
NEXT
END
```

SHELL.APP SHELL-METZNER SORT

```
"SHELL  SORT"    Y=NI
 "Z1"  Y=Y/2
IF Y=Ø THEN RETURN:      REM  Sort complete
Z99=NI-Y
FOR K9=1 TO Z99
  I=K9
   "X2" E2=I+Y
  REM:  In line below change <= to >= for descending order
  IF SA ( I ) <= SA (E2) THEN "X3" ELSE SWAP SA ( I ), SA (E2)
  I=I-Y
  IF I>0 THEN  "X2"
"X3" NEXT K9
GOTO "Z1"
END
```

Note: To sort string arrays instead of numeric arrays add a "$" to the appropriate variables.

Also see "Perpetual Sort" using INDEX$ in the previous chapter.

ARRAY VARIABLES

QUICKAPP QUICK SORT

```
"QUICK  SORT"
REM Improved Quicksort submitted by Johan Brouwer, Luxembourg.
REM Thanks for the submission, Johan.
SP=0:ST(0,0)=0:ST(0,1)=0
ST(0,1)=NI
DO
  L=ST(SP,0): R=ST(SP,1):SP=SP-1
  DO
    LI=L: R1=R: SA=SA((L+R)/2)
    DO
      WHILE SA(LI)< SA
        LI=LI+1
      WEND
      WHILE SA(RI)>SA
        RI=RI-1
      WEND
      LONG IF LI<= RI
        SWAP SA(LI), SA(RI)
        LI=LI+1:RI=RI-1
      END IF
    UNTIL LI>RI
    LONG IF (R-LI) >(RI-L)
      LONG IF L<RI
        SP=SP+1:ST(SP,0)=L: ST(SP,1)=RI
      END IF
      L=LI
    XELSE
      LONG IF LI<R
        SP=SP+1:ST(SP,0)=LI:ST(SP,1)=R
      END IF
      R=RI
    END IF
  UNTIL R<=L
UNTIL SP=-1
RETURN: REM QUICK SORT FINISHED HERE
END
```

Note: To use the QUICK SORT or SHELL SORT with STRING variables, use DEFSTR with the appropriate variables on the first line of the program or put a "$" after all variables that are strings

Be sure to use DEFLEN or DIM to define the length of the string variables. If each element needs 50 characters, then set the length of SA$ to 50. The default is 256 bytes per element for string variables if you do not define the length.

HINTS ON TYPING IN THE PROGRAM: First of all, use line numbers of your own chosing. Indentation in this program is the way ZBasic shows the loops or repetitive parts of the program. You do not need to type in spaces (Make everything flush left). ZBasic will indent the listing automatically when you type LIST or LLIST.

Also see "Perpetual Sort" using INDEX$ in the previous chapter.

ARRAY VARIABLES

ARRAY ELEMENT STORAGE

The following chart illustrates how array elements for each type of variable are stored in memory.

Assumptions:

1. Memory starts at address zero (Ø)

2. Strings were dimmed: DIM 15 VAR$(1,2,2) (Each element uses 16 bytes*)

3. Other arrays dimmed: DIM VAR%(1,2,2), VAR!(1,2,2), VAR#(1,2,2)
   (SINGLE and DOUBLE precision assumed as 6 and 14 digit accuracy.)

4. BASE OPTION of ZERO is assumed.

	RELATIVE ADDRESSES			
Array ELEMENTS	STRING$	INTEGER%	SINGLE Precision!	DOUBLE Precision#
VAR(0,0,0)	00000	00000	00000	00000
VAR(0,0,1)	00016	00002	00004	00008
VAR(0,0,2)	00032	00004	00008	00016
VAR(0,1,0)	00048	00006	00012	00024
VAR(0,1,1)	00064	00008	00016	00032
VAR(0,1,2)	00080	00010	00020	00040
VAR(0,2,0)	00096	00012	00024	00048
VAR(0,2,1)	00112	00014	00028	00056
VAR(0,2,2)	00128	00016	00032	00064
VAR(1,0,0)	00144	00018	00036	00072
VAR(1,0,1)	00160	00020	00040	00080
VAR(1,0,2)	00176	00022	00044	00088
VAR(1,1,0)	00192	00024	00048	00096
VAR(1,1,1)	00208	00026	00052	00104
VAR(1,1,2)	00224	00028	00056	00112
VAR(1,2,0)	00240	00030	00060	00120
VAR(1,2,1)	00256	00032	00064	00128
VAR(1,2,2)	00272	00034	00070	00136

*Length byte adds one extra byte in front of each string element.

Note: Arrays are limited to 32,768 (0-32,767) elements.

LongInteger arrays are also supported. Each element takes four bytes. Macintosh is limited to 2,147,483,647 elements.

MSDOS version 4.0 has a limit of 32,768 (0-32,767) elements for integer arrays and a limit of 65,536 (0-65535) for string and floating point arrays.

GRAPHICS

GRAPHICS

GRAPHICS

Graphics are an extremely important way of communicating ideas. The old adage "A picture is worth a thousand words" is very true. ZBasic offers many powerful screen imaging commands and functions to take advantage of your computer's graphics capabilities.

In addition to having powerful graphic commands, ZBasic defaults to utilizing the same graphic coordinates regardless of the system you happen to be programming on. This is ideal for moving programs from one machine to another without having to make changes to the graphic commands or syntax. Quite a change from the old days.

Definitions of some commonly used graphic terms:

PIXEL The smallest graphic point possible for a given system. Some systems allow you to set the color of a pixel.

RESOLUTION Refers to the number of pixels (dots of light) on a screen. A computer with a resolution of 400 x 400 has 160,000 pixels (high resolution). A computer with 40 x 40 resolution has only 1600 pixels (low resolution).

COORDINATE By giving a horizontal and vertical coordinate you can describe a specific screen location easily. With ZBasic the origin (0,0) is the upper left hand corner of the screen or window.

With a standard device independent coordinate system you can specify a location on the screen without worrying about pixel positions.

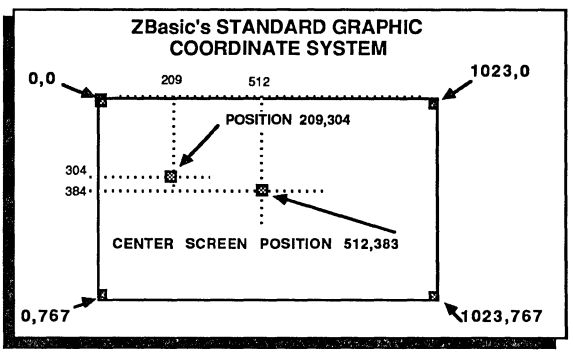
GRAPHICS

ZBASIC'S DEVICE INDEPENDENT GRAPHIC COORDINATE SYSTEM

ZBasic uses a unique DEVICE INDEPENDENT COORDINATE SYSTEM to describe the relative positions on a video screen, instead of a pixel system which describes specific graphic dots on the screen.

The standard coordinate system is **1024** points across (0-1023) by **768** points down (0-767). The width is broader to be in proportion to a normal video monitor.

This approach allows writing graphic programs the same way regardless of a computer's graphic capabilities.



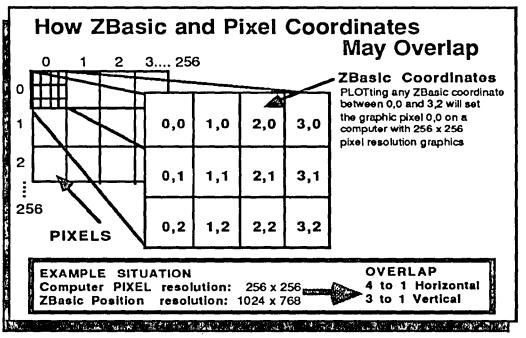
Device independent graphics means the coordinate syntax is the same regardless of the device or type of graphics being used!

The ZBasic approach to graphics makes commands function the same way ***EVEN ON DIFFERENT COMPUTERS!*** ZBasic handles all the transformations needed to match up the ZBasic coordinates to the actual resolution of the computer. This is an ideal way of handling graphics in a standardized way.

On the **MacIntosh** the standard coordinates apply to the current window , not to the screen. **MacIntosh** and **MSDOS** versions of ZBasic have the extra commands; COORDINATE and COORDINATE WINDOW which allow you to set relative coordinates of your own or pixel coordinates, respectively. See the **Apple** appendix for ways of configuring ZBasic to pixel coordinates. Some **Z80** See appendix for specifics.

SCREEN PIXEL versus SCREEN POSITION

It is important to realize that ZBasic's standard coordinate system of 1024 x 768 has a direct relation to the screen, NOT to the actual pixel resolution of the computer being used. It is important not to confuse the pixel coordinate with the position coordinate:

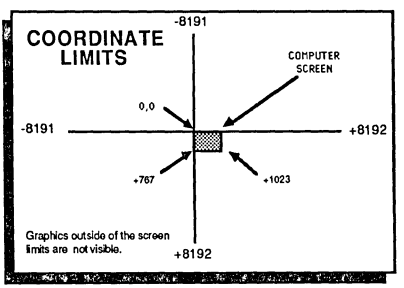


You can see that plotting coordinates; 0,0 through 3,2, sets the same pixel on a screen with 256 x 256 resolution. If the pixel resolution of a computer is 64 x 64 then PLOTting 0,0 or 15, 11 will plot the same pixel (16 to 1 horizontal and 12 to 1 vertical).

Fortunately this information is rarely important. ZBasic takes care of the tedious transformations between different graphic modes and resolutions. Skills learned on one machine may be used on any other machine that uses ZBasic!

OFF SCREEN COORDINATES

ZBasic allows coordinates to be given with graphic commands that are out of bounds of the actual screen coordinates. This allows drawing lines, circles or rectangles off the screen, with only that part of the graphics that are within bounds to be shown on the screen. ZBasic 'clips' the rest of the drawing.



The limits are from -8191 to +8192. Any coordinates given out of this range will cause an overflow and the actual result will be the overflowed amount without generating an error.

GRAPHICS

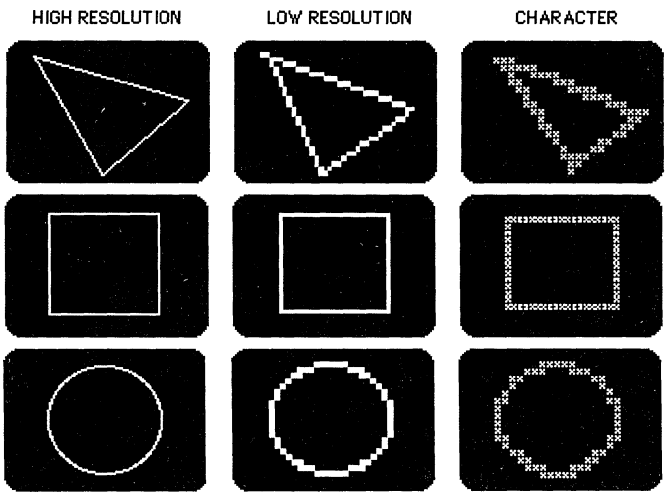
DIFFERENT TYPES OF GRAPHICS

Graphic appearance and quality will depend on the resolution of the computer or terminal you are using. Resolution is the number of graphics pixels on a screen. A computer with a resolution of 40 x 40 has 1600 different pixels. This is low resolution graphics because the graphic points (pixels) are very large.

For computers without graphics, ZBasic will simulate the graphics as closely as possible using an asterisk. The resolution would be the number of characters across by characters down. See MODE.

GRAPHICS TYPE	RESOLUTION
HIGH RESOLUTION	about 200 x 150 or More
LOW RESOLUTION	about 150 x 100 or Less
CHARACTER	TEXT graphics simulation.

A COMPARISON OF LOW AND HIGH RESOLUTION IMAGES



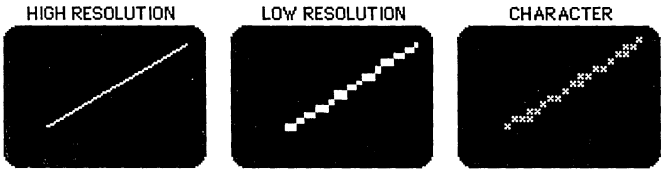
Notice the variation in quality. Programmers porting programs over to other machines should keep the resolution of the target computer in mind when creating programs.

MORE GRAPHIC EXAMPLES AT DIFFERENT RESOLUTIONS

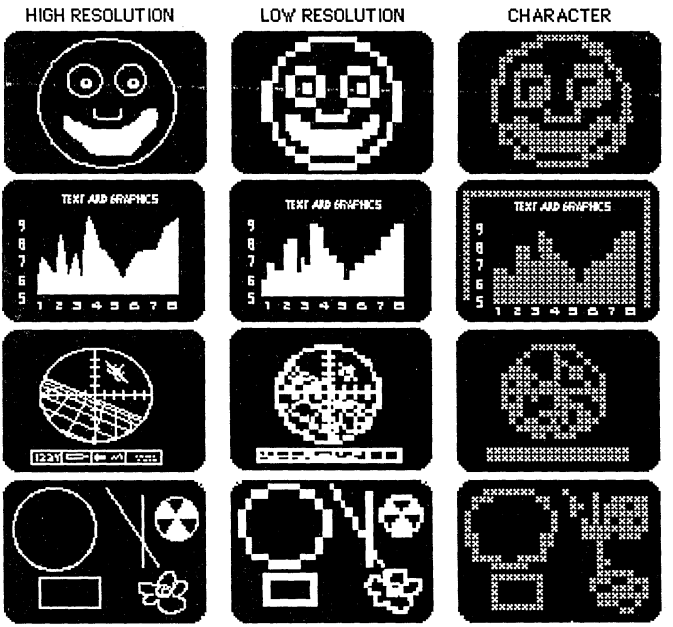
Quality deteriorates as graphic complexity increases and screen resolution decreases , although usually the lower the resolution the faster the execution speed. In this line example you can see the variation of quality.

The ZBasic statement to create all the lines in the first example was the same:

```
PLOT 60,660 TO 1000, 10:
```



Additional examples of more complex graphics forms in different resolutions:



GRAPHICS

MODE

ZBasic offers different modes of text and graphics output depending on hardware and model. The ability to change modes allows you to simulate the output for different machines.
Syntax:

MODE *expression*

The following chart gives the modes for some popular microcomputers, and illustrates how modes are grouped according to resolution.

MODE CHART

Mode number	MSDOS type		APPLE //e, //c		TRS-80 I, III	
	Text	Graphic	Text	Graphic	Text	Graphic
0	40x25	character	40 x 24	character	32x16	character
1	40x25	40x40	none	40x48	64x16	128x48
2	80x25	character	80x24	character	32x16	character
3	80x25	80x25	none	80x48	64x16	128x48
4	80x25	character	40x24	character	32x16	character
5	40x25	320x200	40x24	280x192	64x16	128x48
6	80x25	character	80x24	character	32x16	character
7	80x25	640x200	80x24	560x192	64x16	128x48
8	40x25	character	40x24	character	32x16	640x240?
9	40x25	40x40	Bottom	40x48	64x16	128x48
10	80x25	character	80x24	character	32x16	character
11	80x25	80x25	Bottom	80x48	64x16	128x48
12	80x25	character	80x24	character	32x16	character
13	40x25	320x200	Bottom	280x165	64x16	128x48
14	80x25	character	80x24	character	32x16	character
15	80x25	640x200	Bottom	560x165	64x16	128x48

MACINTOSH	
Text	Graphic
Many Font styles and sizes here!	SEE Macintosh APPENDIX

CP/M-80	
Text	Graphic
Normally 80x24	SEE Z80 APPENDIX

Be sure to read the appropriate appendix for exact mode designations.

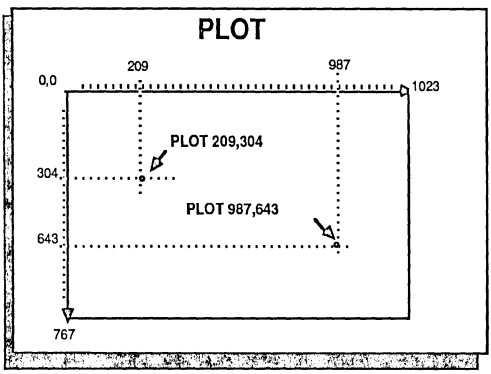
Note: Check your computer appendix for variations.

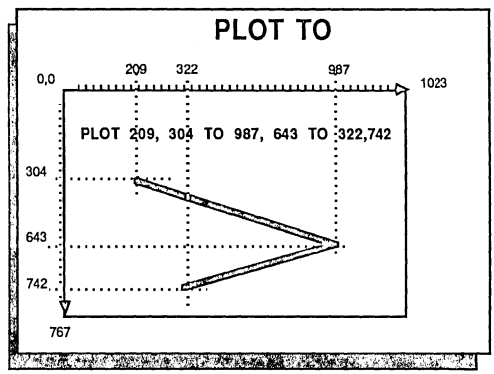
PLOTTING POINTS AND LINES

To set a specific screen position(s) to the current color or to draw lines from one screen position TO another, TO another..., or to draw from the last screen position used (in another ZBasic statement) TO another...

PLOT [**TO**] *horizontal* , *vertical* [**TO** [*horizontal* , *vertical* [**TO**...]]]]

PLOT draws with the last color defined by COLOR. COLOR=0 is the background color of most computers, while COLOR=-1 is the foreground color. If you have a system with a black background, COLOR -1 is white and COLOR 0 is black. See COLOR in this chapter.





As with all other graphic commands, PLOT uses the standard ZBasic coordinates of 1024 x 768 regardless of the computer being used. When TO is used, ZBasic will plot a line from the first position TO the next position, TO the next position...

EXAMPLES OF PLOTTING	RESULT
PLOT 4,5	Turns on the pixel at the graphic position 4 positions over and 5 positions down
PLOT 0,0 TO 1023,767	Plots a line from the upper left corner of the screen down to the lower right corner of the screen.
PLOT TO 12,40	Draws a line from the last position used with the PLOT command TO the point on the screen 12 positions over by 40 positions down.
PLOT 0,0 TO 400,0 TO 0,300 TO 0,0	Plots a triangle in the upper left corner of the screen.

NOTE: All the examples above will plot in the current COLOR.

GRAPHICS

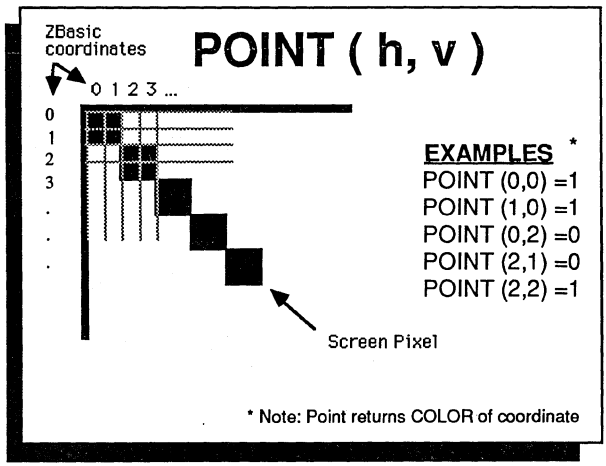
POINT

POINT (*horizontal coordinate, vertical coordinate*)

Returns the COLOR of the pixel at the ZBasic coordinate. Point is available on many computers to inquire about the COLOR of a specific screen graphic position (some computers do not have the capability to "see" pixels).

As with other commands, ZBasic *Device Independent Graphic* coordinates may overlap pixels. The following illustration shows the pixels and color types associated with them.

In this example: Ø=BACKGROUND (WHITE) 1 =FOREGROUND (BLACK)



As with all other ZBasic graphic commands the standard device independent coordinate system of 1024 x 768 is used.

Note: The ZBasic device independent coordinate system specifies positions on the screen, not pixels. See below for ways of setting your system to actual pixel coordinates, if needed.

MacIntosh and **MSDOS** systems can be set to use pixel coordinates with COORDINATE WINDOW. See **Apple** appendix for ways of configuring to pixel coordinates. **Z80** see your hardware technical manual and the Z80 appendix for specifics of your machine.

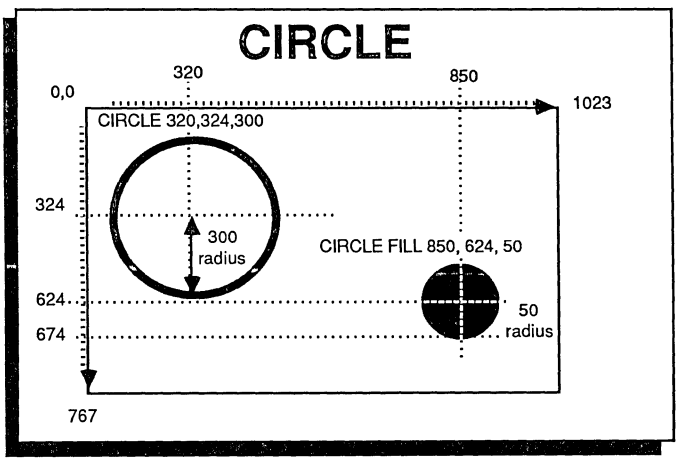
CIRCLE

CIRCLE [FILL] *horizontal , vertical , radius*

CIRCLE draws a circle in the currently defined COLOR and RATIO. COLOR=0 is the background color of most computers, while COLOR=-1 is the foreground color. If you have a system with a black background, COLOR -1 is white and COLOR 0 is black.

See RATIO for ways of changing the shapes of circles. Also see CIRCLE TO and CIRCLE PLOT for creating PIES and ARCS.

If **FILL** is used, the circle will be a solid ball in the current color.



As with all ZBasic graphic commands, the Device Independent Graphic Coordinates of 1024 x 768 are the default.

FILL is taken from PEN pattern; PEN,,,,n. Where n is one of the pen patterns used under the control panel. Quickdraw circles are also available using toolbox calls. See appendix.

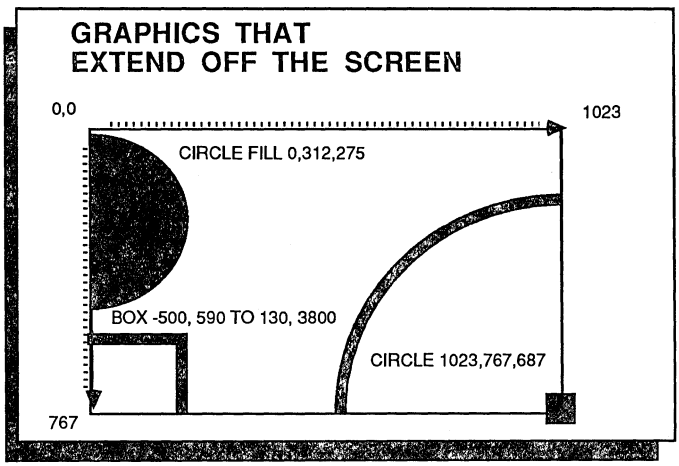
GRAPHICS

GRAPHICS THAT EXTEND OFF THE SCREEN (CLIPPING)

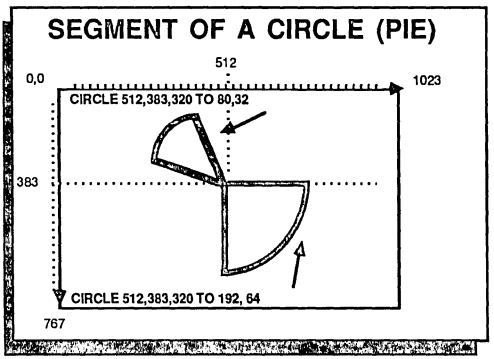
If coordinates are given that exceed the limits of the ZBasic screen coordinates, that part of the image exceeding the limtis will be "CLIPPED".

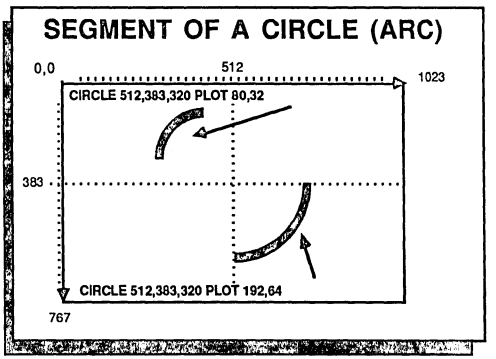
It is still permissible to use these numbers and in many cases it is important to have them available for special effects.

CIRCLE, or other graphic commands like PLOT, BOX, PRINT% etc., with coordinates that are off the screen but are within the limits of -8191 to +8192 are permissible and that part out of range will be "clipped":



As with all ZBasic graphic commands, the Device Independent Coordinates of 1024 x 768 are used.





SEGMENT OF A CIRCLE (PIE)

To draw an enclosed segment of the circumference of a circle (PIE), use this syntax:

CIRCLE *h,v, radius* **TO** *starting BRAD degree, number of BRADs* (counter clockwise)

CIRCLE draws with the last color defined by COLOR. COLOR=0 is the background color of most computers, while COLOR=-1 is the foreground color. If you have a system with a black background, COLOR -1 is white and COLOR 0 is black. See COLOR in this chapter.

SEGMENT OF A CIRCLE (ARC)

To draw a segment of the circumference of a circle (an ARC) use the syntax:

CIRCLE *h, v, radius* **PLOT** *starting BRAD degree, number of BRADs* (counter-clockwise)

CIRCLE draws with the last color defined by COLOR. COLOR=0 is the background color of most computers, while COLOR=-1 is the foreground color. If you have a system with a black background, COLOR -1 is white and COLOR 0 is black. See COLOR in this chapter.

Note: 256 BRADS=360 DEGREES. See the BRAD chart on the next page. As with all ZBasic graphic commands, the standard coordinates of 1024 x 768 are used.

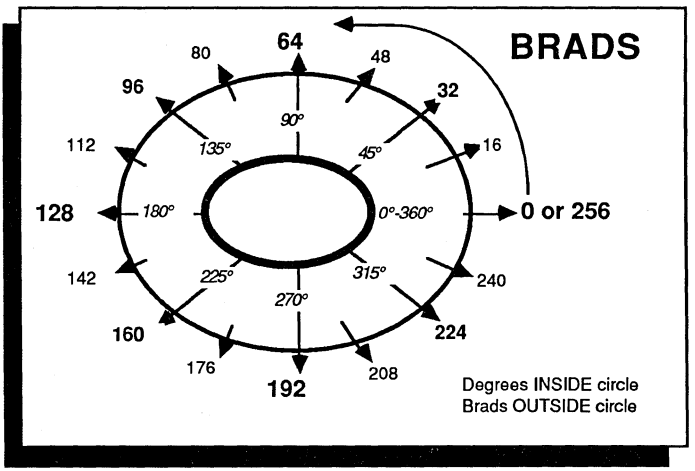
FILL may be used with the CIRCLE FILL x,y,r, TO s,n statement on this version. The FILL pattern is taken from PEN pattern; PEN,,,,n. Where n is one of the pen patterns used under the control panel. Quickdraw arcs are also available using toolbox calls.

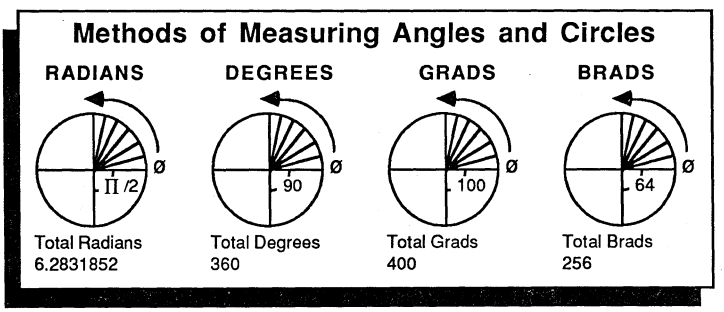
GRAPHICS

BRADS

Brads are used with ZBasic CIRCLE commands to determine a position on the circumference of a circle. Instead of DEGREEs of zero to 359, BRADs range from zero to 255. (Starting at 3 O'clock going counter-clockwise.)

ZBasic™ BRAD CHART





CONVERSIONS FROM ONE TYPE TO ANOTHER

RADIANS=DEGREES*ATN(1)/45
RADIANS=9*GRADS/10
RADIANS=BRADS/40.7436666

DEGREES=RADIANS*45/ATN(1)
DEGREES=BRADS*1.40625
DEGREES=GRAD/63.66197723

GRADS=10 * DEGREES/9
GRADS=RADIANS*63.66197723
GRADS=BRADS*1.5625

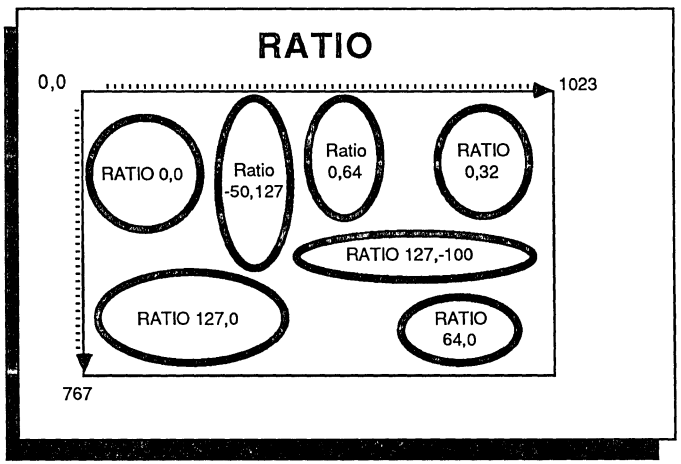
BRADS=DEGREES/1.40625
BRADS=GRADS/1.5625
BRADS=RADIANS*40.743666

Also see USR8 and USR9 for high-speed Integer SIN and COS.

RATIO

ZBasic allows you change the aspect ratio of any CIRCLE, ARC or PIE with the graphic statement RATIO:

RATIO *Width* (-128 thru + 127), *Height* (-128 thru +127)
(See CIRCLE)



Examples:

Ratio settings are executed immediately and all CIRCLE commands will be adjusted to the last ratio.

+127	=	2	times normal
+64	=	1.5	times normal
+32	=	1.25	times normal
Ø	=	Ø	**Normal proportion**
-32	=	.75	times normal
-64	=	.5	times normal
-96	=	.25	times normal
-128	=	0	(no width or height)

Quickdraw circles use box coordinates to set circle shape. See toolbox section of appendix.

GRAPHICS

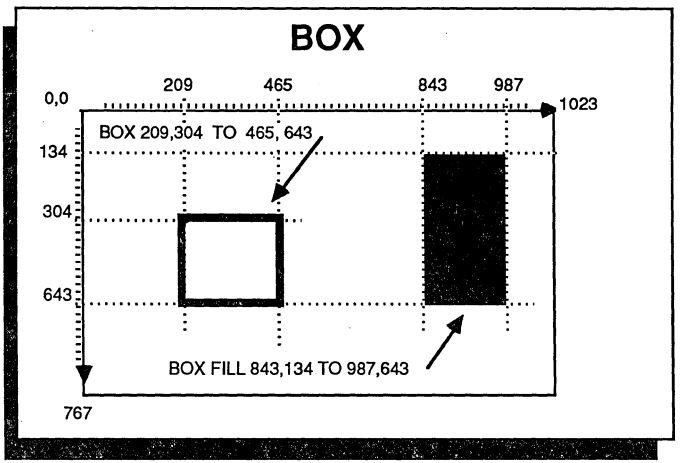
BOX

Box is used for drawing rectangles in the current color. The size of a rectangle is specified by giving the coordinates of opposing corners.

BOX [FILL] *h1, v1* **TO** *h2, v2*

h1, v2	The first corner coordinate of the BOX.
h2, v2	The opposite corner coordinate of the BOX.

The BOX is plotted in the current color. If FILL is used the BOX will be filled with the current COLOR.



As with all ZBasic graphic commands, the device independent coordinates of 1024 x 768 are used. Notice the different quality of BOXes on various computers and different modes.

FILL is taken from PEN pattern; PEN,,,,n. Where n is one of the pen patterns used under the control panel. Quickdraw boxes are also available using toolbox calls. See appendix.

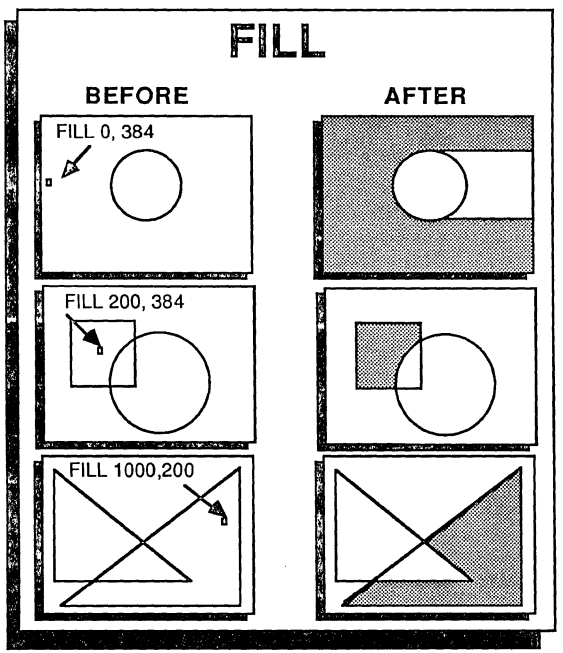
FILL

FILL *Horizontal expression, Vertical expression*

The fill command will fill a screen position from the upper left most position it can reach without finding a color other than the background color, and down to the right and to the left until a non-background color is found.

This command will not function on computers lacking the capability to read screen pixel coordinates. See computer appendix.

Example:



As with all ZBasic graphic commands, the Device Independent Coordinates of 1024 x 768 are used.

Also see **CIRCLE FILL** and **BOX FILL**.

FILL pattern is taken from PEN pattern; PEN,,,,n. Where n is one of the pen patterns used under the control panel. A much faster way to fill screen segments is using Quickdraw FILL with polygons, circles and rectangles. See appendix.

GRAPHICS

COLOR

COLOR is used to signify the color to be used with PLOT, CIRCLE, BOX and FILL. All systems support zero and -1 for background and foreground colors (BLACK and WHITE respectively on most systems).

COLOR [=] *expression*

The following chart represents the color codes for IBM PC and compatible systems with color graphics. Colors codes vary significantly from system to system so check your computer appendix for variations.

IBM PC and Compatible COLOR codes	
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= BRIGHT WHITE

Color intensities will vary depending on the graphics hardware and monitor being used. Check your computer appendix for variations.

While most Macintoshes are black and white, COLOR is useful when printing to the ImageWriter II with a color ribbon. See appendix for details.

CLS, CLSLINE, CLSPAGE

CLS is used to clear the entire screen of graphics or text quickly. Optionally, the text screen may be filled with a specific ASCII character (in most modes). Check your computer appendix for variations.

CLS [*ASCII code:0-255*]

CLS LINE is used to clear a text line of text and graphics from the current cursor position to the end of that line.

CLS LINE

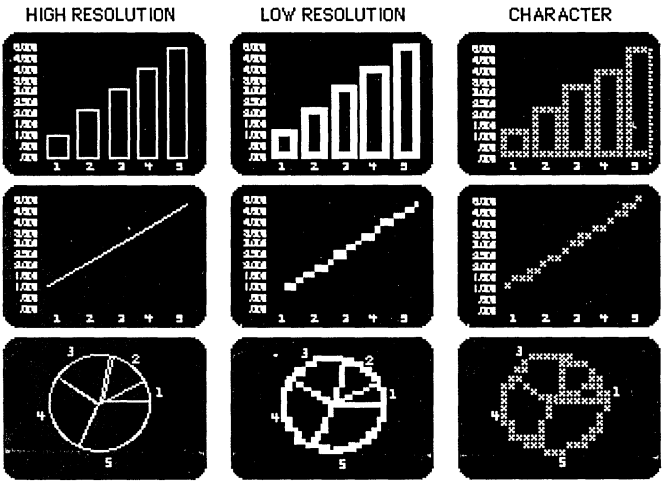
CLS PAGE is used to clear a text screen of text and graphics from the current cursor position to the end of the screen.

CLS PAGE

See Computer Appendix

BUSINESS GRAPHS, CHARTS ETC.

Business graphs and charts are easily accomplished with ZBasic graphics. An added benefit is that the graphs are also easily transported to different computers.



To further assist you in porting graph programs, ZBasic has two text commands that correspond to the graphic position on the screen instead of the text position:

PRINT%*(h,v)*	Prints from the position specified by the ZBasic graphic coordinates.
INPUT%*(h,v)*	Positions the input to be from the graphic position specified by h,v.

The syntax of these commands is the same as PRINT and INPUT. Also see PRINT@.

GRAPHICS

SPECIALIZED GRAPHICS

The Apple, MSDOS, Macintosh and some Z80 versions of ZBasic have some added powerful features for graphics. See the appendix for your version of ZBasic for specific information:

APPLE // GRAPHICS

Double Hi-Res with 16 colors is supported for the Apple //e, //c and //GS with 128K or more. Text and graphic may be integrated on the screen and customizable character sets are also supported. LONG FN's for DRAW, BLOAD and BSAVE are on the master disk.

IBM PC, MSDOS GRAPHICS

Version 4.0 supports most of the graphic modes of IBM PC's and compatibles including; Hercules Monchrome Graphics, Hercules PLUS, Enhanced Graphics Adaptor (EGA), Color Graphics Adaptor (CGA), Monochrome and all other graphics modes.

Also supported are GET and PUT graphic commands, PLOT USING, COORDINATE and COORDINATE WINDOW. See appendix for specifics.

MACINTOSH GRAPHICS

The master disk contains examples of printing and displaying MacPaint graphics and TIFF bit images. Also supported is GET and PUT graphics, PICTURE, TEXT, Apple's QuickDraw and toolbox routines, PEN and many more. See appendix for specifics.

TRS-80, CP/M-80 GRAPHICS

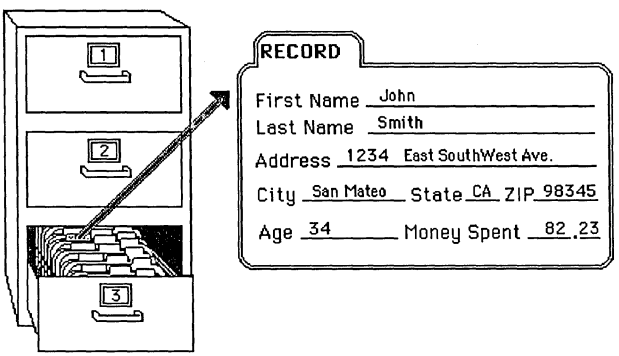
Most TRS-80 graphics are supported including Radio Shack's Hi-Res and Micro-Lab's Hi-Res boards on the Model 4 in MODE 8 and 15 (text and graphic integration is not supported with the Radio Shack Hi-Res board). Hi-Res is not supported on the model one or three.

Because of the diversity of machines for CP/M systems and because of a lack of a common interface, graphics are not supported with CP/M systems (although we have special graphics versions for Kaypro 4 and 10 with graphics capabilities).

FILES

FILE HANDLING

ZBasic file commands are the same on all versions. This section explains file commands and statements. ZBasic file concepts are similar to a file cabinet:



EVERYDAY TERMS

FILE CABINET
Holds files in drawers.

FILE
Contains data for a mail list or inventory control system among other things.

RECORD
One logical part of a file: All the data for Mr. Smith in a mail list (name, address...)

PARTS OF A RECORD
One part of a Record: The address or the City in a mail list record.

ZBASIC TERMS

DISK OPERATING SYSTEM
Holds files on diskettes, cartridges etc.

FILENAME, FILENUMBER
Contains data for a mail list or inventory control system among other things.

RECORD
One logical part of a file: All the data for Mr. Smith in a mail list file (name, address...)

LOCATION
One part of a RECORD: The address in a mail list record or even one character in the address.

FILES

GLOSSARY OF ZBASIC FILE TERMS

DOS: The Disk Operating System is a program residing in a computer's memory which takes care of the actual reading, writing and file control on a storage device such as floppy drives, hard drives, tape backup devices, etc. ZBasic works with the formats and syntax of each disk operating system using its syntax for such things as filenames, drive specs, etc.

FILENAME: Tells ZBasic which file to access. A string constant or variable is used.

FILESPEC: The part of a filename (or some other indicator) that specifies the device, directory or sub-directory a file is on. See your DOS manual for correct filespec syntax.

FILENUMBER: ZBasic may be configured to have from 0 to 99 files OPEN at the same time (if DOS and available memory permit). Filenumbers are used in a program with disk file commands to instruct ZBasic which file is being referred to. For example; if you open a file called "Fred" as number one, when doing file commands you need only refer to file number one, not "Fred". This saves a lot of typing.

RECORD : A record is one segment of a file. A mail list record might include Name, Address, City, State, ZIP, etc. If you want data from a specific record, it is called up using the RECORD command. The first record in a ZBasic file is RECORD 0. There may be up to 65,535 RECORDs in a file.* RECORD #*filenumber, record, location.*

LOCATION: Specifies a location within a record. There may be from 0 to 65,535 locations in a record. Each location in a record can hold one character (1 byte). Location is the second parameter in RECORD; RECORD #*filenumber, record, location.*

SEQUENTIAL METHOD: This is a method of reading a file one element or record at a time, in order ---one after another i.e. 1,2,3... .

RANDOM METHOD: This is the method of reading file items randomly--- out of order. i.e. RECORD 20, 90, 1, 22 ....

FILE POINTER: It is often important to know how to manipulate the file pointer. ZBasic allows you to position the file pointer by using RECORD, and tells you where the file pointer is currently positioned by using REC(filenumber) and LOC(filenumber).

COMPATIBILITY WITH MSBASIC™

Experienced BASIC programmers will like the power and simplicity of ZBasic file commands. For the first time, BASIC file handling commands have been made compatible and portable. All ZBasic disk commands function the same way regardless of the computer being used.

Sequential file commands are very similar. The main difference being that items written with PRINT# should be separated with quoted commas in ZBasic if being read back with INPUT#.

Random file commands have been made simpler, yet just as powerful. Those experienced with MSBASIC file commands should find the conversion painless:

ZBASIC COMMANDS	MSBASIC EQUIVALENTS
READ, WRITE, RECORD	FIELD, GET, PUT, MKI$, CVI, MKS$, CVS, MKD$, CVD, LSET, RSET
PRINT#, INPUT#, LINEINPUT#	PRINT#, INPUT#, LINEINPUT#

FILES

FILE COMMANDS COVERED IN THIS SECTION

This outline gives an overall perspective of file commands available in this section and groups commands in logical order. This section of the manual provides lots of examples and a tutorial for the file commands of ZBasic.

OPENING AND CLOSING FILES
- OPEN
- CLOSE

DELETING OR ERASING FILES
- KILL

RENAMING A FILE
- RENAME

POSITIONING THE FILE POINTER
- RECORD

WRITING TO A FILE
- WRITE#
- PRINT#
- PRINT# , USING
- ROUTE

READING FROM A FILE
- READ#
- INPUT#
- LINEINPUT#

GETTING IMPORTANT FILE INFORMATION
- LOF
- LOC
- REC

Be sure to read the appendix for your computer. Many versions have extra commands that take advantage of a particular system.

FILES

CREATING AND OPENING FILES

OPEN ["O, I or R"], *filenumber*, "*filename*" [,*record length*]

All ZBasic files must be opened before processing.

OPEN "O"
Opens a file for "O"utput only. If the file does not exist, it is created. If it does exist, all data and pointers are erased and it is opened as a new file.

OPEN "I"
Opens a file for "I"nput only. If the file does not exist, a "File Not Found" error is generated for that file number.

OPEN "R"
Opens a "R"andom access file for reading and/or writing. If the file does not exist, it is created. If the file exists, it is opened, as is, for reading or writing.

filenumber
ZBasic may be configured to have from 1 to 99 files open at one time in a program (depending on the DOS and available memory for that computer). Files are assigned numbers so ZBasic knows to which file it is being referred. The original copy of ZBasic is configured to allow up to two open files at a time. If you wish to have more files open, you may configure ZBasic for up to 99 open files. See "Configure".

"filename"
The filename is the name of the file on the disk. Filenames may be string constants or string variables. Filenames may also specify which drive to use. Filename and drive specification syntax is dictated by the disk operating system. See your DOS manual.

record length
Record length is optional. If it is omitted, a record length of 256 characters is assumed. Maximum record length is 65,535 characters, or bytes (check appendix for variations).

EXAMPLES OF OPENING FILES

OPEN "O", 2, "NAMES", 99
Opens filenumber 2 as "NAMES", with a record length of 99 characters, for OUTPUT only. If "NAMES" doesn't exist, a file named "NAMES" is created. If a file called "NAMES" exists, all data and pointers in it are deleted and it is opened as a new file.
OPEN "I",1, A$
Opens filenumber 1 whose filename is the contents of A$, with assumed record length of 256 for INPUT only. If A$ doesn't exist, a "File Not Found" error is generated for filenumber one. See "Disk Error Trapping" for more information.
OPEN "R", 2, "BIGFILE" , 90
Opens filenumber 2 named "BIGFILE", with a record length of 90, for Reading and Writing. If "BIGFILE" doesn't exist it is created.

OPEN"IR", "OR", "RR" for resource forks. OPEN "A" for append also supported. Volumn number is used after record number i.e. OPEN"R",1,"Fred",99, vol%. A number of other enhancements are covered in the appendix.

FILES

CLOSING FILES

CLOSE[# *filenumber* [, *filenumber,...*]]

All files should be closed when processing is finished or before ending a program. *Failure to close files may result in lost data.*

CLOSE without a filenumber closes all open files (STOP and END will also CLOSE all files). It is very important to close all opened files before exiting a program. When a file is closed, the end-of-file-marker is updated and any data in the disk buffer is then written to the disk.

After you close a file, that filenumber may be used again with another OPEN.

DELETING FILES

KILL "*filename*"

Files may be deleted from the disk from within a program or from the editor with the "KILL" command. From the editor the filename must be in quotes on Macintosh and Z80 versions.

Filename is a simplestring and may be represented by a string constant or variable:

```
TRONB
INPUT"FILE TO KILL: ";FILE$
INPUT"ARE YOU SURE? ";A$
IF A$<>"YES" THEN END
KILL  FILE$
END
```

RENAMING FILES

RENAME "*oldfilename*" TO [or comma] "*newfilename*"

Files may be renamed on the disk from within a program or directly using RENAME.

Filenames may be a string constant or variable. Example:

```
TRONB
INPUT"FILE TO RENAME";OLDFILE$
INPUT"NEW NAME: ";NEWFILE$
RENAME OLDFILE$ TO NEWFILE$
```

The TRS-80 Model 1,3 version does not support RENAME.

Macintosh: Both KILL and RENAME also use Volumn number. See appendix for syntax.
MSDOS: CHDIR and Pathnames may be used. APPLE ProDOS: Pathnames may be used.

FILES

WRITING TO A FILE USING PRINT#, WRITE# AND ROUTE#

PRINT#

PRINT # *filenumber,* (*variables, constants or equations*) [;",..."]

PRINT# is used for writing data in TEXT format. It is saved to the disk quite like an image is saved to paper using LPRINT. PRINT# is useful for many things but it is not the fastest way or most efficient way to save data. See WRITE# below. Examples:

PRINT#1, A$;","; C$;","; Z% ;","; X#
Prints A$, C$, Z%, and X#, to filenumber one starting at the current file pointer. A carriage return* is written after the X#. This command stores data the same way it would be printed. Syntax is compatible with older versions of BASIC. The file pointer will point at the location in the file directly following the carriage return.*

PRINT#1,USING "##.##"; 12.1
Formats output to filenumber one starting at the current file pointer (stores 12.10). Functions like PRINT USING.

*Data MUST be separated by a delimiter of a quoted comma or a carriage return if reading data back using INPUT#. Some systems write a carriage return and a linefeed (two bytes).

WRITE#

WRITE [#] *filenumber, variable* [, *variable...*]

WRITE# is used for storing data in condensed format at the fastest speed. WRITE# may only be used with variables and data is read back with the READ# statement. Example:

WRITE#1, A$;10, Z%, K$;2
Writes 10 characters from A$, the value of Z%, and 2 characters from K$ to filenumber one, starting at the current file pointer. In the example; A$;10 stores A$ plus enough spaces, if any, to make up ten characters (or truncates to ten characters if longer).

ROUTE#

ROUTE [#] *device*

ROUTE is used to route output to a specific device. Device numbers are:

Ø	video monitor (default)	**1-99**	DISK filenumber (1-99)
128	PRINTER (same as LPRINT)	**-1** or **-2**	SERIAL port 1 or 2*

Example of routing screen data to a disk file or serial port:

1. Open a file for output (use OPEN "C" and -1 or -2 for serial ports)
2. ROUTE to filenumber or serial port number that was opened.
   All screen PRINT statements will be routed to the device specified.
3. ROUTE Ø (so output goes back to the video)
4. Close the file or port using: CLOSE# n.

* Be sure to see your computer appendix for specifics.

READING FROM A FILE USING INPUT#, LINEINPUT# AND READ#

INPUT#

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

INPUT# is used to read text data from files normally created with PRINT#. The data must be read back in the same format as it was sent with PRINT#. When using PRINT# be sure to separate data items with quoted comma or carriage return delimiters, otherwise data may be read incorrectly or out of sequence. Example:

INPUT#1, A$, C$, Z%, X#
Inputs values from filenumber one from the current RECORD and LOCATION pointer, into A$, C$, Z%, and X#. In this example the data is input which was created using the PRINT# example on the previous page. The file pointer will be pointing to the next location after X#.

LINEINPUT#

LINEINPUT# *filenumber, variable* **(One variable only)**

LINEINPUT# is used primarily for reading text files without the code limitations of INPUT#. Commas, quotes and other many other ASCII characters are read without breaking up the line. It will accept all ASCII codes accept carriage returns or linefeeds. TEXT is read until a carriage return or linefeed is encountered or 255 characters, whichever comes first:

LINEINPUT#5, A$
Inputs a line into A$ from filenumber five from the current file pointer. Accepts all ASCII codes including commas and quotes, except linefeed (chr10) and carriage return (chr 13). Terminates input after a chr 13, chr 10, End-of-file, or 255 characters.

READ#

READ [#] *filenumber, variable* [, *variable...*]

READ# is the counterpart of WRITE#. It is used to read back data created with WRITE# in condensed high-speed format. This is the most efficient way of reading files. Example:

READ#1, A$;10, Z%, K$;2
Reads 10 characters into A$, an integer number into Z%, and 2 characters into K$ from filenumber one, from the current file pointer. The file pointer will be pointing to the location directly following the last character in K$ (includes trailing spaces if string was less than ten).

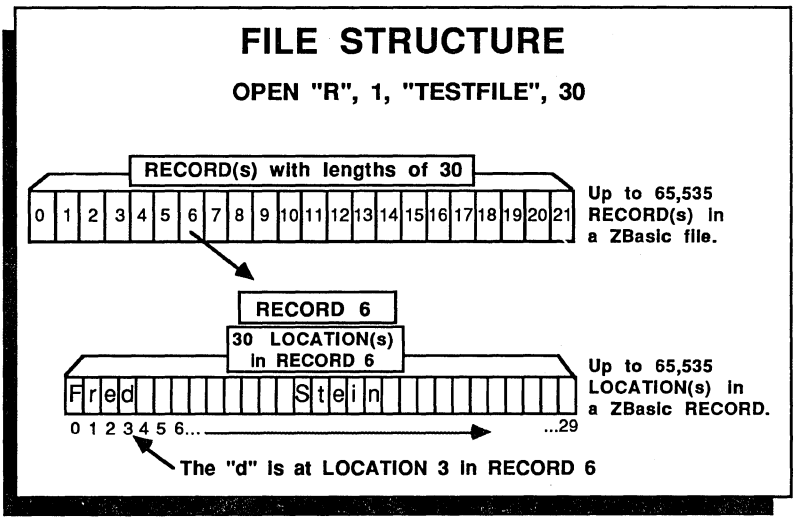
GETTING IMPORTANT INFORMATION ABOUT A SPECIFIC FILE

Syntax	Description
REC(*filenumber* **)**	Returns the current RECORD number location for *filenumber.*
LOC(*filenumber* **)**	Returns the current location within the current RECORD for *filenumber* (the byte offset).
LOF(*filenumber* **)**	Returns the last RECORD number of *filenumber.* If there are one or zero records in the file, LOF will return one. Due to the limitations of some disk operating systems this function is not always exact on some systems. Check the COMPUTER APPENDIX for specifics.

FILES

ZBASIC FILE STRUCTURE

All ZBasic files are a contiguous string of characters and/or numbers (bytes). The order and type of characters or numbers depends on the program that created the file.



In the illustration, the name "Fred Stein" was stored in RECORD six of "TESTFILE". To point to the "d" in FILENUMBER 1, RECORD 6, LOCATION 3 use the syntax:

```
RECORD#1, 6, 3
```

The location within a record is optional, zero is assumed if no location is given. If RECORD 1, 6 had been used (without the 3), the pointer would have been positioned at the "F" in "Fred" which is LOCATION zero.

If RECORD is not used, reading or writing starts from the current pointer position. If a file has just been OPEN(ed), the pointer is at the beginning of the file. (RECORD#n, Ø, Ø)

After each read or write, the file pointer is moved to the next available position in the file .

Macintosh: RECORD length and number of records is 2,147,483,647.

FILES

POSITIONING THE FILE POINTER

RECORD [#] *filenumber*, *RECORD number* [, *LOCATION number*]

To point to any LOCATION in any RECORD in any FILE, use:

RECORD 3, 23, 3	Sets the pointer of filenumber 3 to RECORD 23, LOCATION 3. If RECORD 23 contained "JOHN", then LOCATION 3 of this record would be "N", since zero is significant.
RECORD #3, 23	Sets the pointer for file#3 to location zero in RECORD 23. If RECORD 23 contained JOHN, the character being pointed at would be "J".

RECORD IS OPTIONAL

If the RECORD statement is not used in a program, the pointer will have a starting position of RECORD 0, LOCATION 0 and is automatically incremented to the next position (for reading or writing) depending on the length of the data.

FILE SIZE LIMITATIONS*

The file size limitations for sequential files are either the physical limitations of the storage device or the limit of the Disk Operating system for that computer.

The limitation for Random access files is 65,536 records with each record containing up to 65,536 characters. Maximum file length is 4,294,967,296 characters (although multiple files may be linked to create larger files).

It is important to note that most Disk Operating Systems do not have this capability. Check your DOS manual for maximum file sizes and limitations.

Macintosh: RECORD length and number of records is 2,147,483,647.

CONFIGURING THE NUMBER OF FILES IN A ZBASIC PROGRAM

If the number of files is not configured, ZBasic assumes only 2 files will be used and sets aside only enough memory for two files.

To use more than 2 files, configure ZBasic for the number of files you need under "Configure" .

ZBasic allows the user to configure up to 99 disk files for use in a program at one time (memory and disk operating system permitting). Each type of computer requires a different amount of buffer (memory) space for each file used so check your computer appendix for specifics (usually there are 256--1024 bytes allocated per file; 10 files would require between 2,560-10,240 bytes).

*See computer appendix for variations.

SEQUENTIAL METHOD

SEQUENTIAL METHOD

SEQUENTIAL METHOD

This section covers some of the methods that may used when reading or writing files sequentially. It covers the use of READ, WRITE, PRINT#, INPUT# and LINEINPUT#.

SEQUENTIAL METHOD USING PRINT# AND INPUT#

These two programs demonstrate how to create, write, and read a file with PRINT# and INPUT# using the Sequential Method:

PRINT#

```
OPEN"O",1,"NAMES"
DO: INPUT"Name: "; NAME$
  INPUT "Age:"; AGE
  PRINT#1,  NAME$","AGE
UNTIL NAME$="END"
CLOSE#1:  END
```

INPUT#

```
OPEN"I",1,"NAMES"
DO:  INPUT#1,  NAME$,AGE
  PRINT NAME$","AGE
UNTIL NAME$="END"
CLOSE#1:END
```

Type "END" to finish inputing names in the PRINT# program. The INPUT# program will INPUT the names until "END" is read.

FILE IMAGE CREATED WITH PRINT#

T	o	m	,	2	3	cr lf	H	a	r	r	y	,	4	5	cr lf	G	i	l	d	a	,	1	7	cr lf	K	a	t	h	y	,	1	0	1	cr lf

cr lf Carriage return and sometimes linefeed depending on the Disk Operating System (DOS)

Unless a semi-colon is used after the last data being printed to the disk, the end of each PRINT# statement is marked with a carriage return.

PRINT# USING

USING is used to format the PRINT# data. See "PRINT USING".

COMMAS IN PRINT# AND INPUT#

It is important to remember when using PRINT# with more than one data item, that quoted commas (",") must be used to set delimiters for data being written. If commas are not quoted, they will merely put spaces to the disk (as to the printer) and INPUT# will not be able to discern the breaking points for the data.

SEQUENTIAL METHOD

SEQUENTIAL METHOD USING READ# AND WRITE#

Other commands which may be used to read and write sequential data are READ# and WRITE#. The main difference between READ#--WRITE# and PRINT#--INPUT# is that the latter stores numeric data and string data, much the same way as it appears on a printer; READ# and WRITE# store string and numeric data in a more condensed and predictable format. In most cases this method is also much faster.

VARIABLES MUST BE USED WITH READ# AND WRITE#

READ# and WRITE# require that variables be used for data. Constants or expressions may not be used with these commands except the string length, which may be an expression, constant or variable.

HOW STRINGS ARE STORED USING WRITE#

When using WRITE# or READ# with strings, you must follow the string variable with the length of the string:

WRITE#1, **A$;10, B$;LB** READ#1, **A$;10, B$;LB**

An expression may be used to specify the string length and must be included. When WRITE#ing strings that are shorter than the specified length, ZBasic will add spaces to the string to make it equal to that length. If the string is longer than the length specified, it will be "Chopped off" (If the length of A$ is 20 and you WRITE#1,A$;10, the last 10 characters of A$ *will not* be written to the file).

Normally, you will READ# strings back exactly the same way you WRITE# them. Notice that the spaces become a part of the string when they are READ# back. If you WRITE# A$;5 , and A$="HI" when you READ# A$;5, back, A$ will equal "HI " (three spaces at the end of it). The length of A$ will be 5.

To delete the spaces from the end of a string (A$ in this example), use this statement directly following a READ# statement:

```
WHILE ASC(RIGHT$(A$,1))=32: A$=LEFT$(A$,LEN(A$)-1): WEND
```

You can use READ# and WRITE# using variable length strings as well. See the two format examples on the following pages.

SEQUENTIAL METHOD

READ# AND WRITE# IN CONDENSED NUMBER FORMAT

Numbers are stored in condensed format when using READ# and WRITE#. This is done to conserve disk space AND to make numeric space requirements more predictable. ZBasic automatically reads and writes condensed numbers in this format. Just be sure to read the data in exactly the same order and precision with which it was written. Space requirements by numeric variable type are as follows:

PRECISION	MAXIMUM DIGITS	SPACE REQUIRED
INTEGER	4.3 (±32,767)	2 bytes
SINGLE PRECISION	6 (default)	4 bytes
DOUBLE PRECISION	14 (default)	8 bytes

Since single and double precision may be configured by the user, use this equation to calculate the disk space required if different than above:

(Digits of precision / 2) +1 = number of bytes per variable

LongInteger has 9.2 digits and requires 4 bytes for storage. To calculate the storage needs for Macintosh Double precision; Digits/2+2=space required per variable.

INTEGER NUMBER CONVERSIONS

For those programmers that want to control conversions these commands are available. They are not required with READ and WRITE since these commands do it automatically.

X=**CVI** (simplestring)	Converts the first two bytes of simple-string to integer (X).
A$=**MKI$** (integer)	Converts an integer to a 2 byte string.

SINGLE AND DOUBLE PRECISION NUMBER CONVERSIONS

For those programmers that want to control conversions these commands are available. They are not required with READ and WRITE since these commands do it automatically.

X#=**CVB** (simplestring)	Converts up to the first 8 bytes* of simplestring to an uncondensed double precision equivalent and stores the value in X#. (If string length is less than eight characters, only that many characters will be converted. At least two bytes are needed.)
A$=**MKB$** (X#)	Converts a Double precision number to an 8 byte string.*
X!=**CVB** (simplestring)	Converts the first 4 bytes* of simplestring into a single precision number and stores the value in X! If string length is less than eight characters, only that many characters will be converted. At least two bytes are needed.
A$=**MKB$** (X!)	Converts a single precision number to a 4 byte string.*

*Note: The number of bytes of string space in the conversions depends on the precision set by the user. Use the equation above for calculating the space requirements. ZBasic assumes 8 bytes for double precision and 4 bytes for single precision if the user does not set precision.

To manipulate LongIntegers with MKI$/CVI use DEFSTR LONG. See Macintosh appendix.

SEQUENTIAL METHOD

SEQUENTIAL FILE METHOD USING READ# AND WRITE#

The following programs illustrate how to use READ# and WRITE# using the sequential file method.

USING READ# AND WRITE# WITH SET LENGTH STRINGS

The programs below create and read back a file with the sequential method using READ# and WRITE#. String length is set to 10 characters by the "10" following NAME$. ZBasic adds spaces to a string to make it 10 characters in length, then saves it to the disk.

AGE is assumed to be an integer number since it was not defined and is stored in condensed integer format.

WRITE#

```
OPEN"O",1,"NAMES"
DO: INPUT"Name: "; NAME$
  INPUT"Age:"; AGE
  WRITE#1,NAME$;10,  AGE
UNTIL NAME$="END"
CLOSE#1: END
```

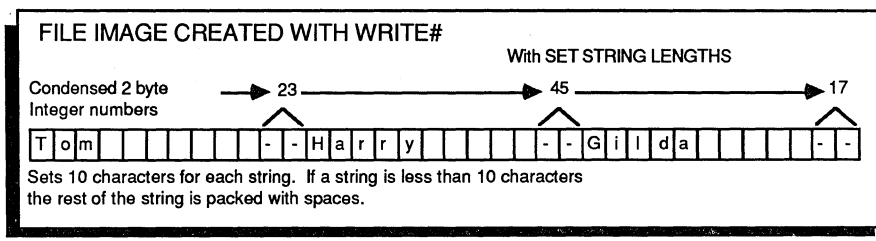
READ#

```
OPEN"I",1,"NAMES"
DO:READ#1,NAME$;10,  AGE
  PRINT NAME$;",";AGE
  A$=LEFT$(NAME$,3)
UNTIL NAME$="END"
CLOSE#1: END
```

Type "END" to finish inputting names for the WRITE# program. The READ# program will READ the names until "END" is encountered.

FIXED STRING LENGTH WRITE#

This illustration shows how strings saved with set lengths appear in a disk file:



The reason the ages 23, 45 and 17 are not shown in the file boxes is because the numbers are stored in condensed format (2 byte integer).

SEQUENTIAL METHOD

USING READ# AND WRITE# with VARIABLE LENGTH STRINGS

READ# and WRITE# offer some benefits over PRINT# and INPUT# in that they will read and write strings with ANY ASCII characters. This includes quotes, commas, carriage returns or any ASCII characters with a code in the range of 0-255. The following programs will save strings in condensed format, using the amount of storage required for each string variable.

WRITE#

```
OPEN"O",1,"NAMES"
DO: INPUT"Name: "; NAME$
  INPUT"Age:"; AGE
  LB$=CHR$(LEN(NAME$))
  WRITE#1,LB$;1
  WRITE#1, NAME$;ASC(LB$),AGE
UNTIL NAME$="END"
LAST$="END":
WRITE#1,LAST$;3:CLOSE#1
END
```

READ#

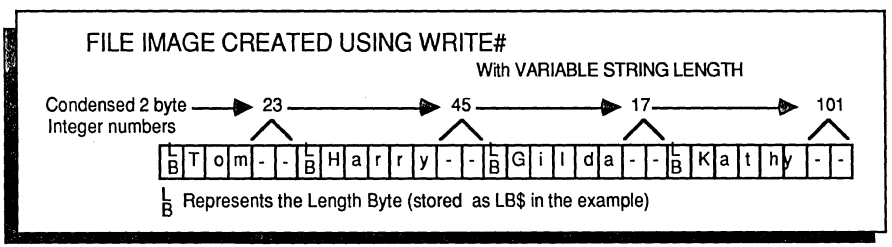
```
OPEN"I",1,"NAMES"
REM
DO: READ#1,LB$;1
  READ#1,  NAME$;ASC(LB$),AGE
  PRINT NAME$","AGE
UNTIL A$="END"
CLOSE#1
END
```

The WRITE# program stores a one byte string called LB$ (for Length Byte). The ASCII of LB$ (a number from 0 to 255) tells us the length of NAME$.

Notice in line 30 (of READ#) that LB$ is read BEFORE NAME$, thus allowing us to read the length of NAME$ first (all data in file handling statements is processed IN-ORDER).

VARIABLE STRING LENGTH WRITE#

This illustration shows how the data is saved to the disk when string data is saved using the variable length method. LB for "Tom" would be 3, LB for "Harry" would be 5, etc...



SEQUENTIAL METHOD

APPENDING DATA TO AN EXISTING FILE CREATED USING THE SEQUENTIAL METHOD

Sometimes it is faster (and easier) to append data to the end of an existing text file, instead of reading the file back in, and then out again.

This may be accomplished by using "R", for random access file when opening the file, and keeping track of the last position in a file using REC(filenumber) and LOC(filenumber) and putting a character 26 at the end of the file .

To append sequentially to a text file created with other programs try using this example program. The key is setting the record length to the right amount. The MS-DOS version uses 128. Other versions will vary.

This example creates a function called: FN Open (f$, F%) and will OPEN the file named f$, with file number f%, for appending. The RECORD pointer will be positioned to the next available space in the file.

To close a file properly for future appending, use the function called FN Close (f$, f%).

```
LONG FN Open (f$,f%):  REM  FN OPEN(f$, f%)
  OPEN "R", f%, f$,128:REM Change 128 to correct# for your DOS
  Filelen%=LOF(f%): NextRec%=FileLen%: NextLoc%=Ø
  LONG IF FileLen%>Ø
    NextRec%=NextRec%-1
    RECORD #f%, NextRec%, NextLoc%
    READ #f%, NextRec$;128:  REM Change this 128 too!
    NextLoc%=INSTR(1,NextRec$,CHR$(26)): REM (zero on Apple)
    IF NextLoc%>0 THEN NextLoc%=NextLoc%-1 ELSE NextRec%=NextRec%+1
  END IF
  RECORD #%f, NextRec%, NextLoc%
END FN


LONG FN Close (f$, f%)
REM TCLOSE the file correctly with an appended chr 26.
PRINT#f%, CHR$(26);
CLOSE#f%
END FN
```

NOTE: This method will work with ASCII text files ONLY!

See Open "A" in the appendix for opening files for Append.

RANDOM METHOD

CREATING FILES USING THE RANDOM ACCESS METHOD

Random access methods are very important in disk file handling. Any data in a file may be stored or retrieved without regard to the other data in the file. A character or field from the last record in a file may be read (or written) without having to read any other records.

A simple example of the Random access method is the following program that reads or writes single characters to any LOCATION in a file:

RANDOM ACCESS EXAMPLE USING A ONE BYTE RECORD LENGTH

```
OPEN   "R" , 1 ,"DATA",1
REM    RECORD LENGTH = 1 character

"Get record number"
DO: INPUT "Record number:  ";RN
  INPUT  "<R>ead,     <W>rite,    <E>nd:  " ; A$
  IF A$="R" GOSUB "Read" ELSE IF A$ = "W" GOSUB "Write"
UNTIL A$="E": CLOSE#1: END


"Write"
INPUT "Enter character: " ; A$
RECORD #1,   RN
WRITE  #1,A$;1  :RETURN

"Read"
RECORD #1,RN :REM  Point at record# RN
READ  #1,A$;1
PRINT" Character in RECORD# "; RN ;" was " ;A$: RETURN
```

To change this program to one that would read or write people's names, merely change the RECORD LENGTH to a larger number and increase the number after the A$ in the READ# and WRITE# statements.

The following pages will demonstrate a more practical use of the Random Access method by creating a mail list program in easy to understand, step by step procedures.

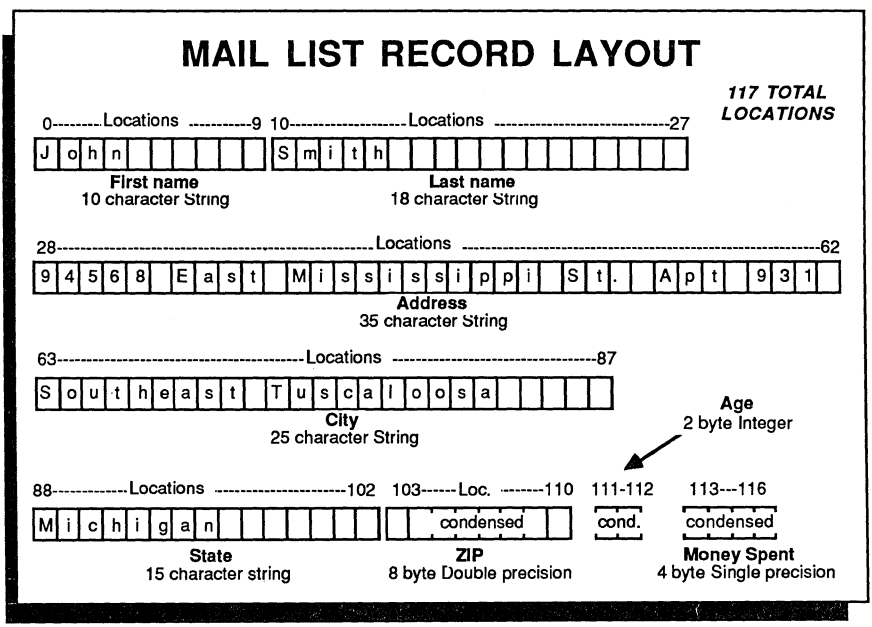
RANDOM METHOD

CREATING A MAIL LIST USING THE RANDOM ACCESS METHOD

This mail list uses: First and Last name, Address, City, State, Zip, Age and Money spent. The first thing to do is calculate the record length for the mail list file. This is done by calculating the space requirements for each field in a RECORD.

FIELD	VARIABLE TYPE	SPACE NEEDED
FIRST NAME	STRING$	10 characters
LAST NAME	STRING$	18 characters
ADDRESS	STRING$	35 characters
CITY	STRING$	25 characters
STATE	STRING$	15 characters
ZIP	DOUBLE PRECISION	8 bytes (holds up to 14 digits)
AGE	INTEGER	2 bytes (Holds up to 32,767)
MONEY SPENT	SINGLE PRECISION	4 bytes (Holds up to 6 digits)
Totals:	**8 VARIABLES**	**117 bytes per RECORD**

The following illustration illustrates how the mail list data is stored within each RECORD. LOCATION numbers are shown by position.



RANDOM METHOD

MAIL LIST PROGRAM

The following program will READ# and WRITE# mail list data as described on previous pages. The names are input from the user and a mail list file record is created for each name.

You will be able to retrieve, print, and search for names in the mail list and, with some simple modifications (like using the sort routines in the ARRAY section of this manual) you will have a complete mail list program ready to use.

EXPLANATIONS OF THE MAIL LIST PROGRAM BY LINE NUMBER

10-21	Asks if you want to create a new file. If you say yes the old data is written over.
22	If old data is being used, the data in RECORD zero is READ to find out how many names are on the disk. NR holds the number of records on the disk.
25-77	Puts a menu on the screen and awaits user input.
80	**"END"** routine. Closes file and exits the program.
100-210	**"ADD"** names to mail list. Gets data from user, checks if OK. If not OK starts over. Note that the spaces in the input statements are for looks only. Space may be omitted.
220	If not OK then redo the input.
230-255	Gets the disk record (DR) from NR. Saves the variables to disk, then increments the number of records. (NR=NR+1) and saves it to disk record zero.
500-590	**PRINT**(s) all the names in the file to the printer. (Change LPRINT to PRINT for screen output).
700-780	**"FIND"** all occurences of LAST NAME or PART of a LAST NAME. To find all the names that start with "G" just type in "G". To find "SMITH" type in "SMITH" or "SMIT" or "SM".
1000-1040	**"READ A MAIL LIST ITEM"** READ(s) RECORD **DR** from the disk into the variables FIRST_NAME$, LAST_NAME$, ADDRESS$, ...
1100-1140	**"WRITE A MAIL LIST ITEM"** WRITES the variables FIRST_NAME$, LAST_NAME$, ADDRESS$, ... out to the RECORD specified by **DR**.

HINTS: Spaces are not important when typing in the program, except between double quotes (if you have set "Spaces required between keywords" they will be required).

RANDOM METHOD

MAIL LIST PROGRAM EXAMPLE

```
0010   CLS
0015   OPEN"R",1,"MAIL",117
0016    INPUT"CREATE A NEW FILE:Y/N";A$: IF A$><"Y" THEN 22
0021   NR=1: RECORD1,0: WRITE#1,NR:REM NR=Number of names in list
0022   RECORD 1,0: READ#1, NR
0025   DO: CLS
0030     PRINT"MAIL LIST PROGRAM"
0040     PRINT"1. Add names to list", "Number of names: ";NR-1
0050     PRINT"2. Print List"
0052     PRINT"3. Find names"
0055     PRINT"4. End"
0060     INPUT@(0,7)"Number: ";ANSWER: IF ANSWER<1 OR ANSWER>4THEN60
0075   ON ANSWER GOSUB "ADD", "PRINT", "FIND"
0077   UNTIL ANSWER=4
0079   :
0080   "END": CLOSE#1: END
0099   :
0100   "ADD"
101    CLS
102    PRINT"MAIL LIST INPUT":  PRINT
0130   INPUT"First Name: ";FIRST_NAME$
0140   INPUT"Last Name: ";LAST_NAME$
0150   INPUT"Address: ";ADDRESS$
0160   INPUT"City: ";CITY$
0170   INPUT"State: ";STATE$
0180   INPUT"ZIP: ";ZIP#
0190   INPUT"AGE: ";AGE%
0200   INPUT"Money Spent:";SPENT!
0210   PRINT
0220   INPUT"Is everything correct?  Y/N: ";A$: IFA$<>"Y"THEN "ADD"
0230   RECORD 1,0:READ#1,NR: DR=NR:  NR=NR+1: REM NR is incremented
0240   GOSUB"WRITE A MAIL LIST ITEM": REM when names added
0250   RECORD 1,0: WRITE#1, NR :      REM Stores records to record zero
0255   RETURN
0260   :
0261   :
0500   "PRINT"
0510   REM  Change LPRINT to PRINT if screen output preferred
0515   RECORD 1,0: READ#1,NR
0520   FOR X=1TO NR-1:  DR=X          :REM  DR=DISK RECORD
0530     GOSUB"READ A MAIL LIST ITEM"
0540     LPRINT FIRST_NAME$;" ";LAST_NAME$
0550     LPRINT ADDRESS$
0560     LPRINT CITY$;",";STATE$;"  ";ZIP#
0570     LPRINT AGE%, "SPENT:"; USING"$###,###.##";SPENT!
0575     LPRINT:IF FLAG=99 RETURN
0580   NEXT
0585   DELAY 3000
0590   RETURN
```

Continued next page

RANDOM METHOD

```
0700   "FIND"
0704   CLS
0705   RECORD 1,0: READ#1, NR
0710   IF NR=1 THEN PRINT "No names to find!":DELAY 999:RETURN
0720   INPUT"NAME TO FIND: ";F$:F$=UCASE$(F$)
0730   FOR X=1 TO NR-1
0740     DR= X:  GOSUB"READ A MAIL LIST ITEM"
0750     T$=UCASE$(LAST_NAME$) :REM CASE must match
0755     IF INSTR(1,T$,F$) THEN FLAG=99: GOSUB 540: FLAG=0
0760   NEXT
0770   INPUT "LOOK FOR ANOTHER?  Y/N:";A$:IFA$="Y" THEN 700
0780   RETURN
0781   :
0782   :
1000   "READ  A  MAIL  LIST  ITEM"
1001   REM:This routine READS RECORD DR
1020   RECORD 1, DR
1030   READ#1, FIRST_NAME$;10, LAST_NAME$;18, ADDRESS$;35,
1035   READ#1, CITY$;25, STATE$;15, ZIP#,  AGE%,  SPENT!
1040   RETURN
1041   :
1042   :
1100   "WRITE  A  MAIL  LIST  ITEM"
1101   REM: This routine WRITES RECORD DR
1110   REM  CALL WITH DR=DISK RECORD NUMBER TO WRITE
1120   RECORD 1, DR
1130   WRITE#1, FIRST_NAME$;10, LAST_NAME$;18, ADDRESS$;35
1135   WRITE#1, CITY$;25, STATE$;15, ZIP#,  AGE%,  SPENT!
1140   RETURN:   END
```

MIXING FILE METHODS

MIXING FILE METHODS

MIXING SEQUENTIAL AND RANDOM FILE METHODS

Since ZBasic stores data as a series of bytes whether sequential methods or random methods are used, these methods may be intermixed.

The following program uses both methods. The program reads files from the mail list program created with the random access method earlier in this chapter.

The second and third lines read the number of records in the file. Then the list is read off the disk sequentially using the DO/UNTIL loop.

To read and print the mail list in sequential order:

```
OPEN"I",1,"MAIL",117
RECORD 1,0:READ#1, NR:REM  Gets number of records to read
RECORD 1,1: REM Set pointer to the first record
REM   Change LPRINT to PRINT if screen output prefered
DO: NR=NR-1: REM  Counts down the number of names
  READ#1, FIRST_NAME$;10, LAST_NAME$;18, ADDRESS$;35,
  CITY$;25, STATE$;15, ZIP#,  AGE%,  SPENT!
  LPRINT FIRST_NAME$;" ";LAST_NAME$
  LPRINT ADDRESS$
  LPRINT CITY$;",";STATE$;"  ";ZIP#
  LPRINT AGE%, "SPENT:"; USING"$###,###.##";SPENT!
 LPRINT
UNTIL NR=1:REM Until the last name is read
CLOSE#1
END
```

The READ#1 after the DO reads the data in. Whenever read or write functions are executed, ZBasic automatically positions the file pointer to the next position.

DISK ERRORS

DISK ERRORS

DISK ERROR MESSAGES

If a disk error occurs while a program is running, ZBasic will print a message something like this:

```
File Not Found Error in File #Ø2
(C)ontinue or (S)top?
```

If you type "S", ZBasic will stop execution of the program and return to the disk operating system (or to the editor if you are in interactive mode).

If you press "C", ZBasic will ignore the disk error and continue with the program. This could destroy disk data!!

The following pages will describe how to "TRAP" disk errors and interpret disk errors which may occur.

END OF FILE CHECKING

Some versions do not have an "END OF FILE" command because some operating systems do not have this capability. Example of END OF FILE checking for some versions:

```
ON ERROR GOSUB 65535:  REM Set for User Error trapping
OPEN"I",1,"DEMO":IF ERROR PRINT ERRMSG$(ERROR):STOP
DO
  LINEINPUT#1,A$
UNTIL ERROR <>0
IF ERROR >< 257 THEN PRINT ERRMSG$(ERROR): STOP
REM 257=EOF Error in filenumber 1(See error messages)
ERROR=0:REM  You MUST reset the ERROR flag.
ON ERROR RETURN:REM Give error checking back to ZBasic
CLOSE#1
```

Note: Many versions have an EOF function. See your appendix for details.

DISK ERRORS

TRAPPING DISK ERRORS

ZBasic provides three functions for disk error trapping:

ON ERROR GOSUB 65535	Gives complete error trapping control to the user. User must check ERROR (If ERROR <>0 then a disk error has ocurred) and take corrective action if any disk errors occur. (Remember to set ERROR=0 after a disk error occurs). ZBasic will not jump to a subroutine when the error occurs. The 65535 is just a dummy number. See the ON ERROR GOSUB line:
ON ERROR GOSUB line	GOSUB to the line number or label specified whenever and wherever, ZBasic encounters a disk error.
ON ERROR RETURN	Gives error handling control back to ZBasic. Disk error messages will be displayed if a disk error occurs.

When you are doing the ERROR trapping it is essential that ERROR be set to zero after an error is encountered (As in line #45 and #1025 in the program example). Failure to set ERROR=0 will cause additional disk errors.

DISK ERROR TRAPPING EXAMPLE

The following program checks to see if a certain data file is there. If disk error 259 occurs (File Not Found error for file #1), a message is printed to insert the correct diskette:

```
 10 ON ERROR GOSUB "CHECK DISK ERROR"
 15 REM Line above Jumps to line 1000 if any disk error occurs
 20 OPEN"I",1,"TEST"
 30 IF ERROR=0 THEN 50
 40 INPUT"Insert Data diskette: press <ENTER>";A$
 45 ERROR=0:REM    You MUST reset ERROR to zero!
 46 GOTO 20 :REM    Check diskette again...
 50 ON ERROR RETURN:REM ZBASIC DOES DISK ERROR MESSAGES NOW...
    .
    .
    .
1000 "CHECK DISK ERROR"
1003 REM    ERROR 259 is "File Not Found Error in File #01"
1005 IF ERROR=259 RETURN
1010 PRINT ERRMSG$(ERROR)     :REM Prints error if not 259
1015 INPUT"(C)ont. or (S)top? ";A$
1020 A$=UCASE$(A$): IFA$<>"C" THEN STOP
1025 ERROR=0:REM      You MUST reset ERROR to zero!
1030 RETURN
```

Note: This method may not work on some Disk Operating Systems (like CP/M). Check your computer appendix for specifics.

DISK ERRORS

DISK ERROR CODES AND MESSAGES

If you wish to do the disk error trapping yourself (using ON ERROR GOSUB), ZBasic will return the ERROR CODE in the reserved variable word **"ERROR"**.

For instance, if a "File not Found Error in file# 2" occurs, then **ERROR** will equal 515. To decode the values of **'ERROR'**, follow this table:

DISK ERROR CODES & MESSAGES

ERROR	ERROR CODE
No Error in File #	Ø
End of File Error in File #	1 (257=file#1, 513=file#2, 769=file#3, etc.)
Disk Full Error in File #	2
File Not Found Error in File #	3
File Not Open Error in File #	4
Bad File Name Error in File #	5
Bad File Number Error in File #	6
Write Only Error in File #	7
Read Only Error in File #	8
Disk Error in File #	9-255

ERROR CODE=ERROR **AND 255**
FILE NUMBER= ERROR **>> 8**

ERROR FUNCTION

ERROR returns a number between Ø and 32,767. IF ERROR does not equal zero then a disk error has occurred. The disk error code of the value returned in ERROR is deciphered by using one of the following equations or statements:

IF ERROR =515 calculate the disk error type by:

ERROR AND 255 =3	File Not Found Error in File #
ERROR >>8 =2	File Number is 2
ERRMSG$(ERROR)=	File Not Found Error in File #02

Also See ERROR and ERRMSG$ in the reference section.

Important Note: To avoid getting the same error again...ALWAYS set ERROR back to zero after an error occurs; ERROR=Ø.

Also see SYSERROR in the Macintosh appendix.

SCREEN AND PRINTER

SCREEN AND PRINTER

SCREEN AND PRINTER

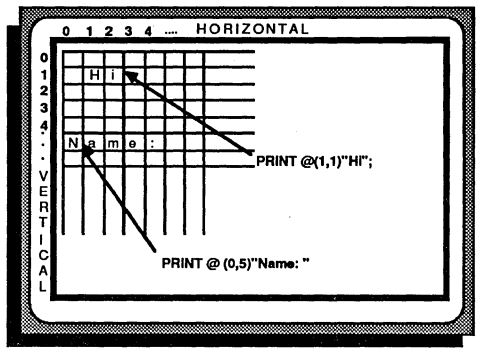
ZBasic has several functions and commands for screen and printer control. PRINT or LPRINT are the most frequently used. The following syntax symbols are used to control the carriage return and TAB for either PRINT or LPRINT:

PRINT SYNTAX	RESULT
Semi-Colon ";"	Suppress Carriage return and linefeed after printing. Subsequent prints will start at the cursor position.
Comma ","	TAB over to the next TAB stop. The default is16: TAB stops are: 16, 32, 48, 64, ... 25 (also see DEF TAB below).
DEF TAB=n	Defines the space between the TAB stops for comma (,). Any number from 1-255. If 10 is used then positions 10, 20, 30, ...250, are designated as TAB stops.

PRINT EXAMPLES	RESULT
PRINT"HI"	Screen PRINT "HI" to the current cursor position and move to the beginning of the next line. <CR>
PRINT "HI";	Screen PRINT "HI" and DON'T move to next line (the semi-colon suppresses the carriage return)
PRINT "HI",	Screen PRINT "HI" and move over to next TAB position.
PRINT TAB(20)"HI"	Print "HI" at the 20th position over from the left or at the current position if past column 20.
PRINT ,"HI"	Print "HI" at the next TAB stop position. See " DEF TAB".
PRINT USING"##.##";23.2	PRINTS 23.20 and moves to the next line. See "USING" in the reference section for further information.
POS(0)	Returns the horizontal cursor position on the screen where the next character will be printed.
POS(1)	Returns horizontal cursor position of the printer where the next character will be printed.

SCREEN AND PRINTER

PRINTING AT A SPECIFIC SCREEN LOCATION



PRINT @(H,V)"HI"	Start printing H characters over horizontally and V lines down vertically from the upper left hand corner of the screen, then move to the beginning of the next line (Use a SEMI-COLON or COMMA to control the carriage return).
PRINT %(Ghoriz, Gvert)	Position the print output to be at the graphics coordinates specified by Ghoriz, GVert (or as close as possible for that computer. Great for easy porting of programs.
CLS [ASCII]	Fill Screen with spaces or optional ASCII characters. (CLS 99 would fill the screen with c's.)
CLS LINE or PAGE	Fill with spaces to the end of the LINE or to the end of the PAGE (screen).
STRING$(Qty, ascii or string)	Used to print STRINGS of charaters. STRING$ (10, "X") prints 10 X's to the current cursor position. STRING$ (25, 32) will print 25 spaces.
SPACE$(n) or SPC(n)	Prints n spaces from current cursor position.
COLOR [=] n	Sets the color of Graphics output and sometimes text. (0= background color, usually black. -1= foreground, usually white).*
MODE [=] n	Sets screen attributes. Some computers allow 80 character across or 40 characters across, etc.. Graphics may also be definable.*
ROUTE byte integer	Used to route output to the screen, printer or disk drive. *

* See Computer Appendix for specifics.

SCREEN AND PRINTER

PRINT %

The PRINT % command functions exactly the same way as PRINT@ *except* the X-Y coordinate specifies a screen graphic position instead of a character position.

Since ZBasic utilizes device independent graphics, this is a handy way of making sure the text goes to the same place on the screen regardless of the computer being used.

Use MODE to set certain character styles for some computers.

Examples:

```
PRINT % (512, 383)     Print to middle of screen
PRINT % (0,0)          Upper left corner of screen
PRINT % (0,767)        Lower left corner of screen
```

Same as the toolbox MOVETO function. ZBasic coordinates unless COORDINATE WINDOW is used.

TYPICAL VIDEO CHARACTER LAYOUTS

Here are some of the typical character layouts for a few of the more popular computers:

COMPUTER	Columns (across)	Rows (down)
IBM PC and compatible	80 or 40	25
APPLE //E, //C	80 or 40	24
TRS-80 Model I, III	64 or 32	16
TRS-80 Model 4, 4p	80 or 40*	24
CP/M-80 computers	80	24
Macintosh	Almost anything...	See appendix

*Will also run TRS-80 models 1, 3 version.

KEYBOARD INPUT

KEYBOARD INPUT

KEYBOARD INPUT

ZBasic utilizes the INPUT and LINEINPUT statements of getting keyboard data from a user. There are many options allowed so that input may be configured for most input types. Parameters may be used together or by themselves in any order. Syntax for INPUT and LINEINPUT:

[LINE]INPUT[;][[@or %] (*horiz,vert*);] [!] [& *n*,] ["*string constant*";] *var*[, *var*[,...]

LINEINPUT	Optional use of INPUT. Allows inputting quotes, commas, and some control characters.
;	A semi-colon directly following "INPUT" disables the carriage return (cursor stays on same line after input).
& *n*,	"&" directly following "INPUT" or semi-colon, sets the limit of input characters to *n*. Length of strings used in INPUT must be one greater than *n*.
!	An exclamation point used with "&" terminates the INPUT when the character limit, defined by "&", is reached, without pressing <ENTER>. If "!" is not used, <ENTER> ends input.
@(*horiz, vert*);	Positions the INPUT message to appear at **character coordinates** *horiz* characters over &*vert* lines down.
%(*horiz, vert*);	Positions the INPUT message to appear at the closest **graphic coordinates** *horiz* pixels over &*vert* pixels down.
"*string constant*";	Prints a message in front of the input.
var [,*var*][,...]	The variable(s) to receive the input. Using more than one variable at a time is allowed except with LINEINPUT.

Important Note: When using strings with INPUT make sure that you define the length of the string at least one character more than will be input.

KEYBOARD INPUT

EXAMPLES OF REGULAR INPUT

EXAMPLE	RESULT
INPUT A$	Wait for input from the keyboard and store the input in A$. Quotes, commas and control characters cannot be input. <ENTER> to finish. A carriage return is generated when input is finished (cursor moves to beginning of next line).
INPUT"NAME: ";A$	Prints "NAME: " before input. A semi-colon must follow the last quote. A carriage return is generated after input (cursor moves to next line).
INPUT; A$	Same as INPUT A$ above, only the semi-colon directly after INPUT disables the carriage return (cursor stays on the same line).

EXAMPLES OF LIMITING THE NUMBER OF CHARACTERS WITH INPUT

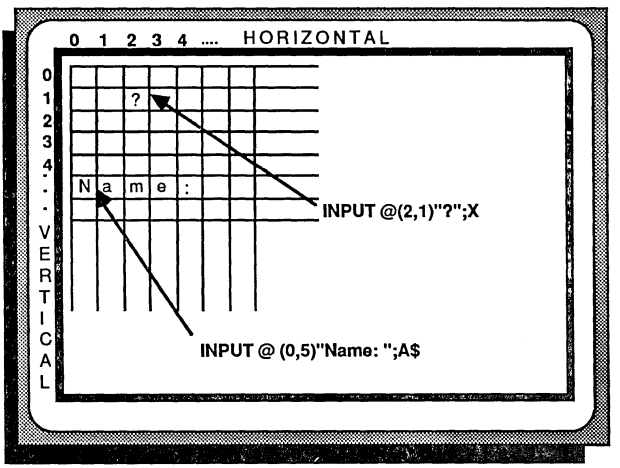
EXAMPLE	RESULT
INPUT &10, A$	Same as INPUT A$ only a maximum of ten characters may be input. (&10) A carriage return is generated after input (cursor moves to the beginning of the next line). The limit of input is set for ALL variables, not each.
INPUT ; &10, I%	Same as INPUT &10, except the SEMI-COLON following INPUT stops the carriage return (cursor stays on line).
INPUT ! &10, A$	Same as INPUT & 10 except INPUT is terminated as soon as 10 characters are typed (or <ENTER> is pressed).
INPUT ;! &10, "NAME: ";A$	Same as INPUT ;&10,A$ except no carriage return is generated (semi-colon). INPUT is terminated after 10 characters(&10 and Exclamation point). and the message "NAME: " is printed first.
LINEINPUT;!&5,"NAME: ";A$	LINEINPUT A$ until 5 characters or <ENTER> is pressed. (no carriage return after <ENTER> or after the 5 characters are input. Accepts commas and quotes.)

Note 1: Wherever INPUT is used, LINEINPUT may be substituted when commas, quotes or some other control characters need to be input (except with multiple variables).

Note 2: If more than one variable is INPUT, commas must be included from the user to separate input. If all the variables are not input, the value of those variables will be null.

In certain cases EDIT FIELD, MENU or BUTTON may be preferable. See appendix.

INPUTTING FROM A SPECIFIC SCREEN LOCATION



INPUT@(H,V); A$	Wait for input at TEXT screen POSITION defined by Horizontal and Vertical coordinates. No "?" is printed. A carriage return is generated.
INPUT %(gH, gV);A$	Input from a graphic coordinate. Syntax is the same as "@". Very useful for maintaining portability without having to worry about different screen widths or character spacing.
INPUT@(H,V);!10,"AMT: ";D#	Prints "AMT:" at screen position H characters over by V characters down. D# is input until 10 characters, or <ENTER>, are typed in, and the input is terminated without generating a carriage return (the cursor DOES NOT go to the beginning of the next line).
INPUT%(H,V);!10,"AMT: ";D#	Prints "AMT:" at Graphic position H positions over by V positions down. D# is input until 10 characters, or <ENTER>, are typed in, and the input is terminated without generating a carriage return (the cursor DOES NOT go to the beginning of the next line).

Note: Replace INPUT with LINEINPUT whenever there is a need to input quotes, commas and control characters (except with multiple variables).

KEYBOARD INPUT

INPUT %

The INPUT % command functions exactly the same way as INPUT@ *except* the X-Y coordinate specifies a screen graphic position instead of a character position.

Since ZBasic utilizes device independent graphics, this is a handy way of making sure the INPUT goes to the same place on the screen regardless of the computer being used.

Use MODE to set certain character styles for some computers.

Examples:

INPUT% (512, 383)	middle of screen
INPUT% (0,0)	upper left corner of screen
INPUT% (0,767)	lower left corner of screen

Although all parameters above function properly, EDIT FIELD, MENU or BUTTON are preferable for getting user input. See appendix.

TYPICAL VIDEO CHARACTER LAYOUTS

Here are some of the typical character layouts for a few of the more popular computers:

COMPUTER	Columns (across)	Rows (down)
IBM PC and compatible	80 or 40	25
APPLE // series	80 or 40	24
TRS-80 Model I, III	64 or 32	16
TRS-80 Model 4, 4p	80 or 40	24
CP/M-80 computers	80	24
Macintosh	Almost anything...	See appendix

KEYBOARD INPUT

INKEY$

Unlike INPUT which must WAIT for characters, INKEY$ can receive characters from the keyboard "on the fly". When INKEY$ is encountered in a program, the keyboard buffer is checked to see if a key has been pressed. For computers with no buffer, the keyboard is checked when the command is encountered. If a key is pressed, INKEY$ returns the key. If no key has been pressed, INKEY$ returns a null string. Examples:

I$=INKEY$	When the program reaches the line with this command on it, ZBasic checks to see if a character is in the input buffer. If a key has been pressed it will be returned in I$. Otherwise I$ will contain nothing (I$ will equal "" or LEN(I$)=zero).
IF INKEY$="S" STOP	If the capital "S" key is pressed the program will stop. Sometimes more appropriate than using TRONB or TRONX for debugging purposes.
DO: UNTIL LEN(INKEY$)	Wait for any key press, then continue
DO: UNTIL LEN(INKEY$)=0	Clears characters out of INKEY$ buffer

Note: TRONX, TRON or TRONB may cause INKEY$ to function improperly!

MacIntosh: If doing EVENT Trapping or any TRON type, the INKEY$ function may operate incorrectly. Use DIALOG(16) instead. See appendix for examples. **MSDOS**: See appendix for special ways of getting function keys (INKEY$ returns two characters).

INKEY$ EXAMPLE

The program below will wait awhile for a key to be pressed. If you make it wait to long, it will complain loudly. If you do press a key, it will tell you which key was pressed. If you press "S" or "s", the program will stop.

```
"Start": CLS
DO
  A$=INKEY$:REM  Check if a key has been pressed
  X=X+1: IF X>3000 THEN GOSUB"YELL FOR INPUT!":REM Timer
UNTIL LEN(A$): REM If a key is pressed then LEN(A$)=1
PRINT "You pressed ";A$
X=0: REM     Reset timer
IF A$="S" OR A$="s" THEN STOP:   REM PRESS "S" to STOP!
GOTO "Start":REM  Go look for another key
:
"YELL FOR INPUT!":REM    This routine complains
PRINT"HURRY UP AND PRESS A KEY!  I'M TIRED OF WAITING"
X=0:REM  Reset Timer
RETURN
```

LOOPS

LOOPS

LOOPS

Loops are sections of a program that repeat over and over again until a condition is met.

Loops are used to make programs easier to read by avoiding IF THEN and GOTO, (although these commands may also be used to loop). ZBasic has a number of ways of looping or executing a routine until a condition is met.

* **FOR, NEXT, STEP**
* **DO, UNTIL**
  WHILE, WEND

* Each of these loop types is executed at least once.

ENTERING OR EXITING LOOPS

ZBasic loops may be entered or exited without ill affects. Some compilers require you to use a loop EXIT statement. This is not required with ZBasic. Just use a GOTO or RETURN to exit as appropriate.

IMPORTANT LOOP REQUIREMENTS

ZBasic requires that each FOR has one, and only one, NEXT. Each WHILE must have one WEND and each DO must have one UNTIL. Otherwise a STRUCTURE error will result when you attempt to RUN the program.

AUTOMATIC INDENTING OF LOOPS

ZBasic automatically indents loops two characters in listings for readability (LIST).

LOOPS

FOR-TO-STEP
NEXT

FOR VAR counter= start expression **TO** end expression [**STEP** expression]
  Program flow...
NEXT [VAR counter]

STEP is an optional part of FOR/NEXT. If STEP is omitted, the step is one. An example of a FOR-NEXT-STEP loop:

```
FOR X=0 TO 20 STEP 2
  PRINT X;
NEXT X
program continues...
```

LINE 1: Begin the loop where X is incremented in STEPs of 2 (0,2,4,6...20)
LINE 2: Prints the value of X each time the loop is executed.
LINE 3: If X => 20 the loop falls through to line 4. X will equal 22 in line 4 of this example program.

FOR-NEXT loops will go through the loop at least once regardless of the values in the FOR instruction. See WHILE-WEND for immediate exiting.

To count backwards in a FOR/NEXT loop set STEP to a negative number.

Note 1: STEP zero will cause an endless loop.

*Note 2: With integer loops, be sure the maximum number is less than 32,767; otherwise an endless loop may occur for some systems. The reason for this is that the sign of the number increments to -32768 after 32767 which restarts the loop all over again! Endless loop example:

```
FOR X%= 1 TO 32767 <--Endless loop!
NEXT X%
```

Note 3: STEP number must stay within the integer range. STEP 32767 would create an endless loop.

Note 4: Unlike most other languages, FOR-NEXT loops may be entered or exited in the middle with no ill effects.

*The same problem arises with four byte integers when the maximum LongInteger number in the FOR loop exceeds 2,147,483,647.

LOOPS

DO
UNTIL

DO
Program flow...
UNTIL conditional expression is TRUE

```
DO
  X=X+2
  PRINT X;
UNTIL  X>19
program continues...
```

LINE 1: Start of the DO loop
LINE 2: Make X=X+2
LINE 3: PRINT the value of X each time the loop is executed.
LINE 4: If X<20 then go back to the beginning of the loop. When X>19 program falls through to the next statement (line 4 in example)

A DO loop will execute at least once. In contrast to WHILE-WEND, which checks the condition at the beginning of the loop, DO-UNTIL checks the condition at the end of the loop. Use WHILE-WEND when you need to check the condition at the beginning.

Note: Unlike most other languages, the loop may be entered or exited in the middle with no ill effects. For instance, in line 2 above, you could used: IF X>10 then RETURN . This would not cause any problems in the program.

LOOPS

WHILE
WEND

WHILE conditional expression
Program flow...
WEND end loop here when condition of WHILE is FALSE

```
WHILE X<20
  X=X+2
  PRINT X;
WEND
program continues...
```

LINE 1: Continue the loop while X is less than 20.
LINE 2: Make X=X+2.
LINE 3: Print the value of X each time the loop is executed.
LINE 4: If X is less than or equal 20 then go back to the WHILE and do the loop again, otherwise continues at the first statement after WEND.

In contrast to DO-UNTIL and FOR-NEXT (which check the condition at the end of a loop), **WHILE-WEND checks the condition at the beginning of the loop and will exit immediately if the condition is not met.**

Note: Unlike most other languages, a WHILE-WEND loop may be entered or exited in the middle with no ill effects. For instance, in line 30 above, you could have used: IF X>10 then RETURN. This would not cause any problems in the program.

FUNCTIONS AND SUBROUTINES

FUNCTIONS AND SUBROUTINES

ZBasic contains some powerful tools for creating re-usable subroutines and appending or inserting them into other ZBasic programs that you create.

APPEND

APPEND is a command that will take an un-line numbered subroutine and insert it anywhere in an existing program. The syntax for the command is APPEND *linenumber or label, filespec.*

To save a subroutine or program without line numbers, use the SAVE+ command. MERGE is available for merging subroutines or programs with line numbers into an existing program.

DEF FN

ZBasic incorporates the DEF FN and FN statements similar to many other BASIC languages. This is very handy for creating functions that may be used like commands in a program.

A function is given a name and may be called and passed variables. FN's save program space. Note that functions may utilize other functions within definitions and program code.

Examples of using DEF FN to create Derived Math functions.

```
DEF FN e# = EXP(1.)
DEF FN Pi# = ATN(1) << 2
DEF FN SEC#(X#) = 1. \ COS (X#)
DEF FN ArcSin#(X#) = ATN (X# \ SQR(1-X# * X#))
DEF FN ArcCos#(X#) = ATN(1.)*2-FN ArcSin#(X#)
```

Examples of program use:

```
PRINT FN Pi#
Angle# = SIN (FN ArcSin#(I#))
PRINT FN ArcCos#(G#)
```

Note: Be sure to define the function at the beginning of the program before attempting to use it otherwise an UN DEF error will result at compile time.

FUNCTIONS AND SUBROUTINES

LONG FN

Included is a sophisticated and powerful multiple line function called LONG FN.

LONG FN allows you to create multi-line functions as large as a subroutine and allows you to pass variables to the routine. This comes in very handy for creating re-usable subroutines that you can insert or APPEND to other programs.

LONG FN is similar to DEF FN except that the function being defined may be many lines long. Use END FN to end the LONG FN subroutine. WARNING: Do not exit a LONG FN except at END FN otherwise system errors may result.

Example of LONG FN to remove trailing spaces from a string:

```
LONG FN RemoveSpace$(x$)
  WHILE ASC(RIGHT$(x$,1)=32
    x$= LEFT$(x$, LEN(x$)-1)
  WEND
END FN= x$
Name$="ANDY       "
PRINT X$, FN RemoveSpace$(Name$)
z$=FN  RemoveSpace$(fred$)
```

Example of a LONG FN for doing a simple matrix multiplication:

```
DIM A%(1000)
LONG FN MatrixMult%(number%, last%)
  FOR temp%= 0 TO last%
    A%(temp%)=A%(temp%)*number%
  NEXT
END FN
A%(0)=1: A%(1)=2:A%(2)=3
FN MatrixMult%(10,3)
PRINT A%(0), A%(1), A%(2)
```

SYNTAX OF DEF FN AND LONG FN NAMES

FN names have the same syntax as variable names. A function that returns a string value should end with a $. A function that returns a double precision value should end with a #.

AUTOMATIC INDENTATION

ZBasic automatically indents that code between a LONG FN and END FN so programs are easier to read.

SAVING FUNCTIONS FOR USE IN OTHER PROGRAMS

To save DEF FN'S or LONG FN's (or any subroutine) for future use, use SAVE+. This saves the subroutine without line numbers so it may be used in other programs by loading with the APPEND command (be sure to avoid line number references and GOTOs in subroutines to make them easily portable).

MORE EXAMPLES OF LONG FN

The following example will check to see if a random file specified by the filename file$ exists. If it does it will open it as a random file. If it does not exist, it will return a disk error.

Remember; with OPEN"R" if the file exists it is opened, if it doesn't exist it is created. You may not want it created in certain circumstances (like if the wrong diskette is in a drive).

```
LONG FN Openfile%(files$, filenum%, reclen%)
  ON ERROR 65535: REM Disk error trapping on
  "Open file"
  OPEN"I",filenum%,file$
  LONG IF ERROR
    LONG IF (ERROR AND 255) <>3
      PRINT@(0,0);"Could not find ";file$;" Check disk drive"
      INPUT"and press <ENTER> when ready";temp%
      ERROR=0: GOTO "Open file"
    END IF
  XELSE
    CLOSE# filenum%
  END IF
ON ERROR RETURN: REM Give error checking back to ZBasic
OPEN"R",filenum%, file$, reclen%
END FN
```

EASY GETKEY FUNCTION

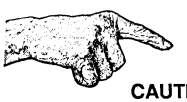
```
LONG FN Getkey$(Key$)
  DO
    Key$=INKEY$
  UNTIL LEN(Key$)
END FN = Key$
```

MACHINE LANGUAGE SUPPORT

MACHINE LANGUAGE SUPPORT

MACHINE LANGUAGE

Occasionally it is important to be able to use machine language programs with your program, whether for speed or to utilize special features of the hardware of that machine. ZBasic incorporates a number of special commands to integrate machine language subroutines into your programs.



CAUTION: Unless you have a working knowledge of the machine language of the source computer and target computer, use extreme caution when porting programs with machine language commands or subroutines.

MACHLG

This statement allows you to put bytes or words directly into your program:

```
CALL LINE "Machlg": END
"Machlg": REM   EXAMPLE ONLY--> DO NOT USE!
MACHLG  10, 23 ,233, 12, 0, B%,  A, 34, 12, &EF
MACHLG  23, 123, 222, 123, 232, GameScore%, &AA
```

Hex, Binary, Octal or Decimal constants, Integer variables, or VARPTR may be used. Be sure to put a machine language RETURN at the end of the routine if using CALL. Be sure you understand the machine language of your computer before using this command.

LINE

This gives you the address of a specific line as it appears in the object code. This allows you to CALL machine language programs starting at specific line numbers or labels. Syntax is

LINE *label*
or **LINE** *line number*

Since the Macintosh is a 16 bit machine, MACHLG code is stored in WORDS not BYTES. The code above would be stored in every other byte. With LINE parentheses are required because it is also a toolbox call i.e. LINE (n).

MACHINE LANGUAGE SUPPORT

CALL

Allows you to CALL a machine language program. The syntax is:

```
CALL address
```

Be sure the routine being called has a RETURN as the last statement if you wish to return control to your program.

If you wish to CALL a machine language subroutine in your program that was made with MACHLG, use CALL LINE *line number* or *label*.

These versions have additional parameter passing capabilities. See appropriate appendix under CALL for specifics.

The ProDOS version provides a special interface to the ProDOS Machine Language Interface (MLI). See appendix for specifics.

DEF USR 0 - 9

Allows you to set up to 10 different machine language user routines. The syntax for using this statement is:

```
DEFUSR digit =address
```

This command may be used to pass parameters or registers. See your computer appendix for the specifics about your computer. There are also default routines. See USR in the reference section.

INTEGER BASE CONVERSIONS

ZBasic makes integer BASE conversions simple. Some of the commands for converting between BASES:

```
BIN$,  &X              UNS$
HEX$,  &H or &         OCT$,   &O
```

See "Numeric Conversions" for specifics.

See DEFSTR LONG for configuring conversions above for LongInteger (and also CVI and MKI$).

MACHINE LANGUAGE SUPPORT

OTHER MACHINE LANGUAGE COMMANDS

Other tools for machine language programmers include powerful PEEK and POKE statements that can work with 8, 16 or 32 bit numbers and BOOLEAN MATH operators.

PEEK, POKE

In addition to the "standard" BYTE PEEK and POKE provided by many versions of BASIC, WORD (16 bit) and LONG (*32 bit) PEEK and POKE are also provided:

```
PEEK          8 BIT          POKE          8 BIT
PEEKWORD     16 BIT          POKEWORD     16 BIT
PEEKLONG     *32 BIT         POKELONG     *32 BIT
```

* Macintosh only at this time.

BINARY/BOOLEAN MATH FUNCTIONS

```
OR                          AND
XOR                         NOT
SHIFT  LEFT                 SHIFT  RIGHT
```

EXP and IMP may be emulated easily. See "Logical Operators" in the "Math" section of the manual.

VARIABLE POINTER

VARPTR (variable) will return the address of that variable.

Macintosh: Remember to use LongIntegers to store the address since Macintosh memory exceeds 65,535 (the limit of regular integers). Also see DEFSTR LONG for defining integer functions to do LongInteger. **MSDOS**: Check appendix for way of determining SEG of variable.

STRUCTURED PROGRAMMING

STRUCTURED PROGRAMMING

STRUCTURE

Much has been said about the difficulty of reading BASIC programs and the so-called spaghetti code created (the program flow is said to resemble the convoluted intertwinings of string spaghetti).

While we believe structure is important , we don't believe that a language should *dictate* how a person should compose a program. This inhibits creativity and may even paint programmers into corners.

Nevertheless, we have provided powerful structure support in ZBasic.

THAT NASTY "GOTO" STATEMENT

The GOTO statement has been classified by many as a programmer's nightmare. If you want programs that are easy to read, do not use this command. If you must use GOTO, do not use line numbers, use labels to make the code easier to follow.

LINE NUMBERS VERSUS LABELS

The standard line editor (command mode) uses line number for three reasons:

1. Remain compatible with older versions of BASIC
2. For the Standard line editor commands
3. To give more easily understandable error messages

To make programs easier to read you should use alphanumeric labels for subroutines or any other area of a program that does a specific function.

It is much easier to follow the flow of a program if GOSUB, GOTO and other branching statements use labels instead of line numbers.

To LIST programs without line numbers use LIST+ . Many versions of ZBasic now use full screen editors that don't require line numbers. See your appendix for specifics.

STRUCTURED PROGRAMMING

INDENTATION OF LOOPS, LONG FN and LONG IF

Some versions of structured languages require that you manually indent nested statements for readability.

ZBasic does all the indenting *automatically!*

Each nested portion of a program will be indented 2 spaces when the program is listed. Program statements like FOR-NEXT, WHILE-WEND, DO-UNTIL, LONG FN, LONG-IF etc. will be indented.

Example using LIST+:

```
LONG FN KillFile(file$)
  PRINT@(0,10);"Erase ";file$;" Y/N";
  DO
    temp$=INKEY$
  UNTIL LEN(temp$)
  LONG IF temp$="y" or temp$="Y"
    KILL temp$
  END IF
END FN
FOR X=1 TO 100
  DO : G=G+1
     WHILE X<95
       PRINT "HELLO"
     LONG IF J< 4
       J=J+1
     END IF
     WEND
   UNTIL G >= 3.5
 NEXT X
```

MULTIPLE LINE STATEMENTS

ZBasic allows putting more than one statement on a line with ":" (colon). While this is handy for many reasons, over-use of this capability can make a program line very difficult to understand.

UNSTRUCTURED

```
10*FORX=1TO100:DO:G=G+1:PRINT G:UNTILG=99:NEXT
```

STRUCTURED

```
 FOR X = 1 TO 100
   DO : G=G+1
     PRINT G
   FOR V=1 TO 20:NEXT
   UNTIL G=99
 NEXT X
*FOR V=1 TO 20:NEXT
```

*Note: An asterisk will appear at the beginning of a line containing a complete loop if that line is not already indented. In that case the line will be un-indented two spaces (as in the examples above).

STRUCTURED PROGRAMMING

SPACES BETWEEN WORDS

To make code more readable, you should insert spaces between words, variables and commands, just as you do when writing in English. While ZBasic does not care if spaces are used (unless you configure ZBasic to require spaces), it is a good practice to insert spaces at appropriate places to make reading the program easier.

Hard to Read `IFX=93*24THENGOSUB"SUB56"ELSEEND`
Easier to Read `IF X=93*24 THEN GOSUB "SUB56" ELSE END`

VARIABLE NAMES

To make code more readable, use logical words for variables.

Hard to Read `B=OP+I`
Easier to Read `Balance = Old_Principle + Interest`

ZBasic allows variable name lengths up to the length of a line , but only the first 15 characters in the name are significant. Do not use spaces or symbols to separate words in a name, use underlines; Building_Principle, Freds_House.

Keywords may not be used in variable names unless they are in lowercase and "Convert to Uppercase" is "NO" (this is the default). Also see next paragraph.

INCLUDING KEYWORDS IN VARIABLES

To allow keyword in variables configure ZBasic for; "Spaces Required after Keywords" (not available on all systems). See "Configure".

HOW CASE AFFECTS VARIABLE NAMES

To make the variable "FRED" and "fred" the same variable configure ZBasic for "Convert to Uppercase". See "Configure".

GLOBAL VERSUS LOCAL VARIABLES

Programmers familiar with LOCAL variables in PASCAL and some other languages can structure their variable names to approximate this in ZBasic. (All ZBasic variables are global.)

GLOBAL variables should start with a capital letter.

LOCAL variables should start with lowercase. Many programmers also use (and re-use) variables like temp$ or local$ for local variables.

STRUCTURED PROGRAMMING

DEFINING FUNCTIONS

Use DEF FN or LONG FN to define functions and then call that function by name. This is easy reading for people trying to decipher your programs. It saves program space as well. FN names have the same definition as variable names. Passing values to functions in variables is also very easy.

LONG FN may be used when a function the size of a subroutine is needed. One FN may call previously defined functions.

LOADING PREVIOUSLY CREATED SUBROUTINES

To insert subroutines you have used in previous programs, use the APPEND command. This will append (or insert) a program saved with SAVE+ (a non-line numbered program or subroutine), into the current program starting at the line number you specify; APPEND *linenumber or label* filename

Be sure to avoid the use of line numbers or GOTO statements in your subroutine to make them more easily portable.

If using variables that are to be considered LOCAL, we recommend keeping those variables all lowercase characters to differentiate them from GLOBAL variables (all ZBasic variables are GLOBAL).

Sometimes LONG FN may be more appropriate for re-usable subroutines.

LISTINGS WITHOUT LINE NUMBERS

To make program listings easier to read, use `LIST+` or `LLIST+` to list a program without line numbers.

ZBasic automatically indents nested statements with LIST for even more readability.

Macintosh: Listings can be sent to the Screen, LaserWriter or ImageWriter without linenumbers and with keywords boldfaced by using `LLIST+*`.
MSDOS: Screen listings with highlighted keywords and no linenumbers are accomplished with `LIST+*` (no printer support for highlighted keywords).

LONG IF

For more control of the IF statement, ZBasic provides LONG IF for improved readability and power.

UNSTRUCTURED

```
10 IFX=ZTHENY=10+H:G=G+Y:F=F+RELSEGOSUB122:T=T-1
```

STRUCTURED

```
LONG IF X=Z
  Y=10+H
  G=G+Y
  F=F+R
XELSE
  GOSUB"READ"
  T=T-1
END IF
```

UNSTRUCTURED

```
10 FORI=-3TO3:PRINT"I= ";I:IF I> THEN IF I>-3 AND I<3
PRINT I;">O",ELSEPRINT"Inner If False":GOTO 30
20 *PRINT I;"<=0",:X=-4:DO:X=X+1:PRINT"X=";X:UNTILX=I
30 NEXT I
```

STRUCTURED

```
FOR I = -3 TO 3: PRINT "I= ";I
  LONG IF I > 0
     LONG IF I > -3 AND I < 3
       PRINT I;"> 0",
     XELSE
       PRINT "Inner LONG IF false"
     END IF
  XELSE
     PRINT I;"<= 0",
     X = -4
     DO : X=X+1
       PRINT"X=";X
     UNTIL X=I
  ENDIF
NEXT I
```

Important Note: Any loops enclosed in LONG IF structures must be completely contained within the LONG IF, END IF construct.

The Macintosh and IBM versions also support SELECT CASE, a structured, multi-conditional LONG IF statement. See appendices for syntax.

DEBUGGING TOOLS

DEBUGGING TOOLS

DEBUGGING TOOLS

To get programs running bug-free in the shortest amount of time, ZBasic has incorporated some powerful error catching tools.

TRON Display program flow

Turns on the line trace statement. As the program is running, ZBasic will display the line number where the program is being executed on the screen.

Also see TRON 128 for sending the line numbers to the printer so the display is not affected.

TRONS Single Step

SINGLE STEP line trace debugging. Allows you to single step through that part of a program. To activate single step mode press CTRL Z. To single step press any key. To return to regular mode press CTRL Z again. To single step and display line numbers use TRONS:TRON. Note: CTRL S and CTRL Z will function during any TRON type.

TRONB Check for <BREAK> key

Sets a break point on that line and all the following lines of that program (until a TROFF is encountered) . As each line is executed, the program will check if CTRL C or <BREAK> is being pressed.

If <BREAK> is pressed, the program will return to the edit mode (the operating system if RUN* was used). Without a break point the program will not respond to the <BREAK> key. No line numbers are displayed unless TRON was also used.

BREAK ON is often preferable as a check for <COMMAND PERIOD>. See appendix.

DEBUGGING TOOLS

TRONX Check for <BREAK> on that line only

Sets a break point only on that line. If CTRL C or <BREAK> is pressed as that line is executed, the program will return to the edit mode (if interactive) or to the operating system.

TROFF Disable all TRON modes

Turns off TRON, TRONB, TRONX and TRONS. Line number display and <BREAK> points will be disabled in the program flow following this statement.

ARRAY BOUNDS CHECKING

Set "Check Array Bounds" to "YES" when configuring ZBasic to make sure you do not exceed DIM limits. This is a *RUN TIME* error check and is very important for use during the debug phase.

Exceeding array limits could cause overwriting of other variables and faulty data.

After you have finished debugging your program, disable this function since it will slow execution speed and increase program size.

STRING LENGTH CHECKING (not all versions; check your appendix)

Set "String Length Checking" to "YES" when configuring ZBasic to make sure you do not exceed defined string length limits. This is a *RUN TIME* error check and is very important for use during the debug phase.

Exceeding string lengths could cause overwriting of other variables and/or faulty data.

After you have finished debugging your program, you may wish to disable this function since it will slow execution speed and increase program size.

COMPILE TIME ERROR CHECKING

ZBasic compile time error messages help you pinpoint the cause of the problem immediately by highlighting the error on the line and printing a descriptive message instead of an error number.

Unlike BASIC interpreters, ZBasic will not execute a program with syntax errors in it. If the program compiles without an error you can be sure it is at least free of syntax errors.

DISK ERROR CHECKING

ZBasic gives the programmer a choice of trapping disk errors themselves or letting ZBasic display the disk error. See "Disk Error Trapping" for more information.

PORTING PROGRAMS

PORTING PROGRAMS

Porting means taking a program from one computer and moving it to another computer of different type or model. As from an Apple to an IBM.

Because most ZBasic commands contained in the reference section of this manual (except USR, OUT, INP, PEEK, POKE, VARPTR, CALL and MACHLG) function the same way, it is very easy to move the source code from one machine to another.

The following pages will describe some of the problems and solutions of porting programs.

OBJECT CODE AND SOURCE CODE

There are two separate types of programs created with ZBasic and you should understand the differences.

SOURCE CODE This is the text part of a program you type into the computer and looks like the commands and functions you see in this manual. In order to turn SOURCE CODE into OBJECT CODE, ZBasic *compiles* it when you type RUN (or RUN* or RUN+).

OBJECT CODE The OBJECT CODE is what ZBasic creates from the SOURCE CODE after you type RUN. Object code is specific to a certain machine. i.e. an IBM PC uses an 8088 CPU and and Apple // uses a 6502 CPU. The ZBASIC OBJECT CODE for each of these machines is different and cannot be ported. Port the SOURCE CODE to the target machine and then recompile it into the OBJECT CODE of that computer.

FILE COMMANDS

ZBasic file commands work almost exactly the same way from one computer to the next. The areas to be aware of when porting code from one machine to another are covered in the following two paragraphs.

PORTING PROGRAMS

DISK CAPACITIES

Make sure the target machine has enough storage space to accommodate the program and program files being ported.

COMMON DRIVE CAPACITIES

IBM PC, XT, jr.	5.25"	320K-360K
	3.50"	780K
IBM PC AT	5.25	360K
	3.50"	780K
	variable density	1200K
Apple // series	5.25"	143K
	3.50"	800K
Macintosh	single sided	400K
Macintosh Plus	double sided	800K
Other:		
SSSD	5.25"	80K
SSSD	8.00"	200-500K
SSDD	5.25"	160K
DSDD	5.25"	320K
DSDD	8.00"	600-2000K

SSSD: Single sided Single density
SSDD: Single sided Double density
DSDD: Double sided Double density

FILESPECS

ZBasic filenames/filespecs work within the limitations of the disk operating system. When porting programs make sure the filespecs are corrected. For instance; if porting a program from a TRS-80 Model 3 to an IBM PC, you must change all references to a file like; Fred:1 to A:Fred

Some computers cannot do RENAME or EOF. Others are incapable of certain DISK ERRORS. Be sure to study the DOS manual of the target machine for variations.

MEMORY

Memory is another area of importance when porting programs from one machine to another.

Porting from smaller machines to machines with larger memory should not be a problem, as long as other hardware is similar. Programs from TRS-80 MODEL I, III, 4, Apple //e and //c and CP/M 80 machines should port over to an IBM PC or Macintosh with little or no changes.

Porting a large program (128K or more) from a larger machine like an IBM PC or Macintosh to a smaller machine will require a number of memory saving measures covered in the following paragraphs:

CHAINING PROGRAMS TOGETHER

If a 128K program is being moved to a 64K system, you will have to split it up into two or more separate programs and CHAIN them together. Since ZBasic allows sharing variables while chaining, this should solve most problems.

PORTING PROGRAMS

CHECK STRING MEMORY ALLOCATION

ZBasic allows the user to change the size of strings. Since some programmers on larger machines may not be concerned with creating efficient code or keeping variable memory use down, check if string size has been set. Setting string size from the 256 byte default to 32 or 64 will reduce string variable memory requirements dramatically.

See DEFLEN, DIM and "String Variables" in this manual for more information about allocating memory for strings.

QUOTED STRINGS

Excessive use of quoted strings often occurs on larger computers because there is so much free memory. Shortening quoted strings may save memory. Also see ZBasic PSTR$ function for an extremely efficient way of utilizing string constants in DATA statements and in regular statements.

EFFICIENT CODE

Careful examination of source code may uncover ways to decrease code size by making repeated commands into subroutines or FN's, or just cleaning up inefficiencies.

RAM DISKS

Some smaller computers allow the use of RAM disks. The Apple // ProDOS version for example, allows RAM disks up to 8 megabytes, while program and variable size are limited to 40K-50K. Utilizing a RAM disk to store indices, large arrays or whatever is nearly as fast as having that data in direct memory.

USE DISK INSTEAD OF MEMORY

If very large arrays or indices have been used in a large program you may have to store and access them from disk in a random file. This is slower than RAM access but is usually quite acceptable on most systems.

TEXT WIDTHS

Some computers have only 64 or 40 characters across the screen or 16 rows down the screen. You may have to adjust the program to accommodate this.

You should think about using the **PRINT% or INPUT%** commands if you plan on porting programs often. PRINT% puts text at ZBasics' device independent graphic coordinates, not text column/row coordinates. This makes porting programs much simpler. Here are some typical character layouts :

COMPUTER	Columns (across)	Rows (down)
IBM PC and compatible	80 or 40	25
Apple // series	80 or 40	24
TRS-80 Model I, III	64 or 32	16
TRS-80 Model 4, 4p	80, 64 or 32	16 or 24
CP/M-80 (typical)	80	24
Macintosh	Almost anything...	See appendix

PORTING PROGRAMS

CHARACTER SETS

Screen and printer characters vary from one computer to the next. Check the ASCII chart in the owners manuals to see the differences. (Most between 32-127 are the same.)

KEYBOARD, JOYSTICK AND MOUSE

Keyboards vary from computer to computer so be sure the target computer has the same keys available. If not, make changes in the program to use other keys.

Joystick and MOUSE devices vary considerably. Test the controls on the target computer and make adjustments for the hardware.

SOUND

Sound tone may vary from machine to machine. Check program and make any adjustments needed. Some machines may not have this capability at all.

DEVICE INDEPENDENT GRAPHICS

ZBasic makes use of very powerful and simple graphic commands that work the same way regardless of the graphic capabilities of the target computer (or lack of).

You will have to determine if the graphic hardware on the target computer is of sufficient quality to display the graphics of your program. Note: Colors and grey levels may have to be adjusted. Here are some of the typical graphic types available for some major computers:

COMPUTER	Horizontal x Vertical pixels
IBM PC and compatibles	CGA: 640x200 (3 color) or 320x200 (8 color)
	EGA: 640x348 (many colors)
	HERCULES and HERCULES PLUS: 720x348
	MDA: 80x25 (text simulation)
Apple][,][+, //e, //c, //GS	Hi-Res 280x192 (6 color)
Apple //e, //c, //GS	Double Hi-Res 560x192 (16 color)
Macintosh	512x340 (larger monitors also supported)
TRS-80 Model I, III	128x48
TRS-80 Model 4, 4p	160x72
	RS and Micro-Lab's hi-res boards 640x240
CP/M-80 (typical)	80x24 (text simulation)
KAYPRO with graphics	160x100

MACHINE DEPENDENT SUBROUTINES

If the program being ported contains machine language subroutines, you will need to rewrite those routines in the machine language of the target computer. Watch out for:

DEFUSR	USR	OUT	INP
MACHLG	LINE	CALL	
PEEK	PEEKWORD	PEEKLONG	
POKE	POKEWORD	POKELONG	

Unless you completely understand the machine language of both the target and source computer, ***use extreme caution*** when porting programs with these commands.

PORTING PROGRAMS

MACHINE SPECIFIC COMMANDS

In order to take advantage of unique or special features of some computers, ZBasic offers special commands that will not work or function on others. Be sure the program you are porting contains only commands from the reference section of this manual.

Special ZBasic commands may have to be rewritten for the target computer.

Be sure to read the ZBasic appendices for both the Target ***and*** Source computers. They will explain in detail the special commands for each system (you must purchase a version of ZBasic for each computer you wish to compile from).

METHODS OF TRANSFERRING SOURCE CODE FROM ONE MACHINE TO ANOTHER

Telephone Modem Transfer
Transfer files using a Modem and simple communications software routines like the ones under OPEN"C" in the main reference section of this manual.

Serial (RS-232) Transfer
Transfer files over the Serial (RS-232) ports of the two computers using a good communication software package like *Crosstalk* or *SmartCom. Crosstalk* is available at computer or software stores nationally.

Diskette File Transfer Utility Programs
Use Diskette file transfer utility programs like *Uniform* or *Interchange.* These programs will convert a file from one disk format, like from a TRS-80 diskette, to another disk format, like MS-DOS or CP/M. These programs are available from computer or software dealers nationally.

Re-type the Program
Type the program into the other computers. This may be acceptable for small programs but you will save plenty of time by using one of the options above.

See OPEN"C" in the reference section for a ZBasic terminal routine that may be used to transfer files.

Important Note: Always transfer files in ASCII. Tokens are not necessarily the same from one version of ZBasic to another and from old versions to newer versions on the same machine.

CONVERTING OLD PROGRAMS

CONVERTING OLD PROGRAMS

CONVERTING PROGRAMS WRITTEN IN OTHER VERSIONS OF BASIC

ZBasic is a very powerful and improved version of BASIC. Many of the traditional BASIC commands have been retained to make conversion as easy as possible. Nevertheless, ZBasic is not 100% compatible with every BASIC. You will have to make some changes to your old programs if you wish to convert them to ZBasic.

If file and graphic handling are not used, conversion will normally be very simple. If files or graphics are used the conversion will take a little more thinking. The following pages will give you important insights into making the conversion process as easy as possible.

The following pages will give you some ideas about converting your older BASIC programs. Following the paragraphs step-by-step will make conversion much easier.

SAVE YOUR OLD BASIC PROGRAM AS ASCII OR TEXT

Save your old BASIC program in ASCII or TEXT format so it can be loaded into ZBasic. ZBasic tokens are different from other BASIC tokens so loading them without first converting them to ASCII will make programs loaded look like random control codes or the wrong commands (if the program will load at all).

See the owners manual for the older BASIC to determine how to save in ASCII or TEXT format for your computer. The typical syntax is; *SAVE "filename",A* .

Note: When upgrading to newer versions of ZBasic, programs may have to be saved in ASCII in the older version before loading into the newer version since tokens may have changed.

CONVERTING OLD PROGRAMS

CONFIGURING ZBASIC TO MAKE CONVERSIONS A LOT EASIER

ZBasic has been configured to give you maximum performance. When converting older BASIC programs this can be a problem. Often they are configured for ease of use instead of performance. ZBasic allows you to configure options so that converting your programs is simpler. Setting some of the options below will also make ZBasic more like the BASIC you may be used to (like MSBASIC and BASICA).

Be sure to see "Configure" in the main reference section and in your appendix for details about other ways of configuring ZBasic.

To solve many of the problems encountered in converting we suggest setting the following options when converting other programs. Be sure to set these options BEFORE LOADING your program:

CONFIGURE OPTION	SET TO
1. Double precision digits of accuracy	6 or 8
2. Single Precision Accuracy	4 or 6
3. Array bounds checking Y/N	Y
4. Default Variable type <S>ingle, <D>ouble, <I>nteger	S
5. Convert to Uppercase Y/N	Y
*6. Optimize expressions for Integer Y/N	N
*7. Spaces Required between Keywords Y/N	Y

1. Since ZBasic does all floating point operations in double precision, it is important to configure ZBasic for the speed and accuracy that you need. In most cases the configuration above will be suitable (but not in all cases). If you wish disk files and memory requirements to be the same as MSBASIC leave the digits of accuracy at 14 and 6 as they take up 8 bytes of Double and 4 bytes for single (the same as MSBASIC).

2. Set to two digits less than Double precision.

3. Sets array bounds checking to give runtime errors. Set to "N" when your program is debugged.

4. Set to Single (S) if you want code to be most like other BASICs. We highly recommend you set it to Integer if possible. Integer will often increase program speeds 10 to 100 times.

5. Setting allows variables like "Fred" and "FRED" to be the same variable. If you want CASE to be significant, do not change the configuration.

6. ZBasic gives you two options for deciding how expressions may be evaluated. ZBasic defaults to optimizing expressions for Integer to get the fastest and smallest code. Most other languages do not. Set to "N" for easier conversions. See "Math" for explanation of ZBasic options for expression evaluations.

7. Some BASICs allow using keywords in variables (like INTEREST) . To allow this, spaces or other non-variable type characters are required around keywords. Set this for easier conversion in most cases (especially IBM PC and Macintosh BASIC type programs).

*Note: Not available on all versions of ZBasic.

CONVERTING OLD PROGRAMS

CONVERTING RANDOM FILES

ZBasic incorporates FIELD, LSET, MKI$, MKS$, MKD$, CVI, CVS, and CVD into the READ and WRITE statements saving the programmer a lot of time. RECORD is used instead of GET and PUT for positioning the file pointer.

The OPEN and CLOSE statements are the same for both BASICs except for MSBASIC use of OPEN FOR RANDOM type. This is changed easily.

ZBASIC statements	MSBASIC equivalents
OPEN"R"	OPEN"R" or OPEN FOR RANDOM
READ, WRITE, RECORD	FIELD, GET, PUT, LSET, RSET, CVS, CVD, MKS$, MKD$, CVI, MKI$

Note: While ZBasic also supports MKI$, CVI and MKB$, CVB, they are not necessary for use in Random files since ZBasic's READ and WRITE automatically store and retrieve numeric data in the most compact format (ZBasic's MKI$, CVI, MKB$ and CVB are most useful for condensing numbers for other reasons). Since ZBasic allows using any variable type in READ and WRITE statements, the user is not faced with complicated conversions of strings-to-numbers and numbers-to-strings.

CONVERTING SEQUENTIAL FILES

Most ZBasic Sequential file commands are very similar or the same to MSBASIC.

ZBASIC statements	MSBASIC equivalents
OPEN"I" or OPEN"O"	OPEN"I", OPEN"O" or OPEN"A" or OPEN FOR INPUT,
OPEN"A" some versions	OUTPUT or APPEND some versions
EOF(n) some versions	EOF(n) some versions
LINEINPUT, INPUT, PRINT	LINEINPUT, INPUT, PRINT

Note: The biggest difference when converting sequential file statements is that ZBasic's PRINT# statements should have quoted commas:

MSBASIC: `PRINT#1, A$,B$,C$` or `PRINT#1, A$ B$ C$`
ZBASIC: `PRINT#1, A$","B$","C$`

DISK ERROR TRAPPING

ZBASIC statement	MSBASIC equivalent
ON ERROR GOSUB	ON ERROR GOSUB

Read "ON ERROR" and "Disk Error Trapping" in this manual for detailed information. ZBasic error codes are much different from MSBASIC.

Important Note: ZBasic does not necessarily store data in disk files in the same way or format as other versions of BASIC. You may have to convert existing BASIC files to ZBasic format.

CONVERTING OLD PROGRAMS

CONVERTING GRAPHIC COMMANDS

ZBasic's Device Independent Graphics are very powerful and simple to understand. Conversion should be painless in most cases:

ZBASIC GRAPHICS	MSBASIC equivalent
PLOT	LINE, PSET, PRESET
CIRCLE	CIRCLE
BOX	LINE (with parameters)
COLOR	COLOR (PSET, PRESET black and white)
MODE	SCREEN
POINT	POINT
GET, PUT (some systems)	GET, PUT (some systems)
RATIO	aspect parameter of CIRCLE
FILL	PAINT
PLOT USING	DRAW

ZBasic defaults to a relative coordinate system of 1024x768. This system does not pertain to pixels but to imaginary positions on the screen. Most older versions of BASIC use pixel coordinates.

MacIntosh and **MSDOS**: Use COORDINATE WINDOW at the beginning of program to set a program to pixel coordinates. **Apple:** See appendix for ways of using POKE to set system to pixel coordinates.

LOOP PAIRS

All ZBasic FOR-NEXT, WHILE-WEND and DO-UNTIL loops must have matching pairs. Some BASIC interpreters allow the program to have two NEXTs for one FOR, or two WENDs for one WHILE. Since ZBasic is a compiler it will not allow this. A STRUCTURE ERROR will be generated when you compile a program with unmatched LOOP pairs.

Another way to find unmatched pairs is to LIST a program. Since ZBasic automatically indents loops, just read back from the end of the LISTing, looking for the extra indent, to find the unmatched statement.

COMPLEX STRINGS

Complex strings may have to be converted to simple strings (some machines).

Improper B$=LEFT$(Right$(A$,12), 13)
Proper B$=RIGHT$(A$,12) : B$=LEFT$(B$,13)

IF-THEN statements may have only one level of complex string.

Improper IF B$=LEFT$(A$,5) THEN GOSUB "END"
Proper C$=LEFT$(A$,5): IF B$=C$ THEN GOSUB "END"

CONVERTING OLD PROGRAMS

LONG LINES

Multiple statement lines with over 253-256 characters (depending on computer) will automatically be shortened by ZBasic when loading. That part of the line longer than 253 will be added to a new line number. Most programs do not have lines of that length.

TIMING LOOPS

Timing loops may have to be lengthened to make up for ZBasic's faster execution time. For some BASIC Languages a FOR-NEXT loop of 1000 would take second or two. (About 1/1000 of a second in ZBasic!) Replace these types of delay loops with the ZBasic DELAY statement.

STRING MEMORY ALLOCATION

Important Note: ZBasic assumes a 255 character length for every string and string array element and allocates 256 bytes for each (255+1 for length byte) unless string length is defined with DIM or DEF LEN.

Many versions of BASIC, like BASICA™, MSBASIC™, APPLESOFT™ and others, allocate string memory as a program needs.

While this may seem efficient on the surface, immense amounts of time are wasted in "String Garbage Collection". Garbage Collection is what happens when your program suddenly stops and hangs up for two or three minutes while BASIC rearranges strings in memory . This makes this method unusable for most serious programming.

HOW DIMMING STRING ARRAYS AFFECT PROGRAM CONVERSION

MSBASIC™:	CLEAR 10000	Sets aside 10,000 bytes for ALL strings
	DIM A$(1000)	Uses memory allocated with CLEAR plus 3-8 byte pointers per element.
ZBasic™:	DIM A$(1000)	256,256 bytes allocated (1001x256)
ZBasic™:	DIM 10 A$(1000)	10,010 bytes allocated (1001x10)

Many BASICs use CLEAR to set aside memory for strings. *Each string in ZBasic is allocated memory at compile time.*

A problem you may encounter while converting: Out of Memory Error from DIMension statements, like the ones above (just define the length of the string elements).

ZBasic allows you to define the length of any string with DEFLEN or DIM statements. Check the string requirements of the program you wish to convert and set the lengths accordingly.

If you have large string arrays that must have elements with wide ranging lengths (constantly changing from zero to 255 characters), use ZBasic's special **INDEX$** string array. Like other BASIC's CLEAR is used to set aside memory for this array (no "Garbage collecting" here either).

See **INDEX$, DEFLEN, DIM** and **"String Variables"** for more information.

CONVERTING OLD PROGRAMS

OTHER INFORMATION

Check your appendix for more information about converting programs.

A good resource for information about converting from one version of BASIC to another is David Lien's *"The BASIC Handbook"*.

CONVERTING OLD COMMANDS

Some BASIC(s) have commands that may be converted over quickly using a word processing program. Simply load the BASIC ASCII file into the word processor and use the FIND and REPLACE commands. (You may also use ZBasic FIND command if you choose.)

A good example would be converting Applesoft™'s HOME commands into ZBasic's CLS command. Have the word processor FIND all occurrences of HOME and change them to CLS.

If you don't have a word processor try using this simple ZBasic convert program to change commands in a BASIC file quickly (file MUST have been saved in ASCII using **SAVE***).

SINGLE COMMAND CONVERSION PROGRAM

```
ON ERROR GOSUB "DISK ERROR": REM Trap Disk Error
INPUT"Command to Change:";Old$
INPUT"Change to:";New$
CLS: PRINT" Changing File....One Minute please"
OLDFILE$="oldfile":NEWFILE$="newfile": REM <-- Change to correct filenames
OPEN"I",1, OLDFILE$
OPEN"O",2, NEWFILE$
WHILE ERROR=0
  LINEINPUT#1, Line$
  DO
     Line$=LEFT$(Line$,I-1)+New$+RIGHT$(Line$,LEN(Line$)-I+1+LEN(Old$))
    I=INSTR(1, Line$, Old$)
  UNTIL I=0
  PRINT#2, Line$
WEND
"Done Changing"
ERROR=0
CLOSE
PRINT "All '";Old$;"' have been converted to '";New$;"'"
INPUT"Rename OLD file? Y/N: ";A$:  A$=UCASE$(A$)
IF A$="Y" THEN KILL OLDFILE$
RENAME "NEWFILE" TO OLDFILE$
END
"DISK ERROR"
PRINT ERRMSG$(ERROR)
CLOSE: STOP
```

Important: Practice on a dummy file until you are sure the program is working properly.

KEYWORDS

ZBASIC™

STANDARD STATEMENTS, FUNCTIONS AND OPERATORS

ABS
AND
ASC
ATN
BIN$
BOX
CALL
CHR$
CIRCLE
CLEAR
CLOSE
CLS
COLOR
COS
CVB
CVI
DATA
DATE$
DEF
DEFDBL
DEFINT
DEFSNG
DEFSTR
DELAY
DIM
DO
ELSE
END
ERRMSG$
ERROR
EXP
FILL
FIX
FN
FOR
FRAC
GOSUB
GOTO
HEX$
IF
INDEX$
INDEXF
INKEY$
INP
INPUT
INSTR
INT
KILL
LEFT$
LEN
LET
LINE
LOC
LOCATE
LOF
LOG
LONG
LPRINT
MACHLG
MAYBE
MEM
MID$
MKB$
MKI$
MOD
MODE
MOUSE
NEXT
NOT
OCT$
ON
OPEN
OR
OUT
PAGE
PEEK
PLOT
POINT
POKE
POS
PRINT
PSTR$
RANDOM
RATIO
READ
REC
RECORD
REM
RENAME
RESTORE
RETURN
RIGHT$
RND
ROUTE
RUN
SGN
SIN
SOUND
SPACE$
SPC
SQR
STEP
STOP
STR$
STRING$
SWAP
TAB
TAN
THEN
TIME$
TO
TROFF
TRON
UCASE$
UNS$
UNTIL
USING
USR
VAL
VARPTR
WEND
WHILE
WIDTH
WORD
WRITE
XELSE
XOR

IMPORTANT: See your computer appendix for other keywords that pertain to your version of ZBasic. Most versions of ZBasic offer more and also use two-word keywords like LONG FN, POKE WORD etc.

KEYWORDS

STANDARD COMMANDS

APPEND	HELP	NEW
AUTO	LIST, L or period "."	QUIT
DELETE or DEL	LLIST	RENUM
DIR	LOAD	RUN
EDIT, E or comma ","	MEM	SAVE
FIND or semicolon ";"	MERGE	

STANDARD REFERENCE

STANDARD REFERENCE GLOSSARY

This reference section is an alphabetical listing of the "Standard ZBasic Commands". The following paragraphs describe the information layout and syntax of this section.

TYPE OF INFORMATION CONTAINED IN THIS REFERENCE SECTION

function	Returns a value; used wherever an expression is used
statement	Executed by itself
command	Used from the standard line editor mode; EDIT, SAVE...
operator	Like AND, OR, XOR or NOT

COMPATIBLE COMMANDS

BLACK BAR	Indicates the command is the same on all versions of ZBasic.
SPECKLED BAR	Indicates the command may not be available on all versions. Check to see if your system does not support that command.

PAGE LAYOUT

The pages are layed out in the same way. Whenever possible descriptions are kept to one page. The header has the command type and description. Paragraph layout is:

FORMAT	Correct syntax for that statement, function or command
DEFINITION	Definition or explanation of usage
EXAMPLE	Program example or direct example of usage. Note that linenumbers are usually omitted. Add linenumbers if needed.
REMARK	Other information of importance and usually a reference to other related sections that will aid the understanding of that item.

IMPORTANT NOTE ABOUT DIVIDE

ZBasic compiles divide symbols based on configuration.

If the default expression evaluator; "Optimize Expressions as Integer?" is YES;
/=integer divide \=floating point divide
If the expression evaluator; "Optimize Expressions as Integer?" is NO;
/= floating point divide \=integer divide
See "Configure" and "Converting Old Programs" and "Math expressions" for more information about the options offered for expression types and how they are evaluated.

continued next page...

STANDARD REFERENCE

CROSS REFERENCE

These commands work the same way on almost every version of ZBasic. There is an extensive cross-reference to other commands and how a command works on specific machines. The reference section uses a computer icon to bring attention to a specific version of ZBasic. The following icons are used:

Apple // DOS 3.3 and ProDOS versions.

MSDOS and IBM PC and compatible versions.

The Macintosh versions (all except the 128K machine).

Z80 machines; Amstrad, CP/M-80 2.x and higher, Kaypro Graphics versions and TRS-80 model 1, 3 and 4 versions.

SYNTAX GLOSSARY

GLOSSARY	DEFINITION
RUN or COMMAND	What follows is program or command output.
[*brackets*]	Items within the brackets are optional (may be omitted)
{ A | B | C }	Any one of A, B or C may be used
... *repeats*	Three periods following items indicates a repeating sequence
`Courier text`	Something you type in, a program example, or program output
expression or *expr*	Numeric: Any; including integer and floating point
byte expression	Numeric: 0-255
word expression	Numeric: 0 to 65,535 or ±32,767
long expression	Numeric: 0 to 4,294,966,293 or ±2,147,483,647
variable or *var*	Any Variable
var$, var%, var&, var!, var#	String, integer, LongInteger, single or double precision variable types, respectively
"string"	Quoted strings (string contants)
simplestring or *string*	String variable, string contant, BIN$, CHR$, HEX$, INDEX$, OCT$, PSTR$, STR$, SPACE$, STRING$ or UNS$.
filenumber	File number: An expression 1-99. See "Configure"
filename	A legal filename for that operating system filename
filespec	Drive or storage volume specifier
line	A line number from 0 to 65,534 or a "label"
number	Requires a number. No variable or expression allowed
var name	A valid variable name

Be sure to take note when you see this hand. It is pointing out important information about using that command. If there is the message **"Important Note"** with the hand it is even more critical that you read the notes.

function **ABS**

FORMAT **ABS** (*expression*)

DEFINITION Returns the absolute value of an expression . The absolute value is the value without regard to the sign (negative, zero or positive).

The result of ABS will always be a positive number or zero.

EXAMPLE

```
A=-15: B=15
PRINT  ABS(A), ABS(B), ABS(-555)
X=ABS(0)
PRINT X

RUN

15, 15, 555
0
```

REMARK The SGN function will return the sign of an *expression* .

AND operator

FORMAT *expression$_1$* **AND** *expression$_2$*

DEFINITION Used to determine if BOTH conditions are true. If both expression1 AND expression2 are true (non-zero), the result is true. Returns -1 for true, 0 for false. See AND truth table below.

Also used to compare bits in binary number operations. 1 AND 1 return a 1, all other combinations of 0's and 1's produce 0. See truth tables below.

EXAMPLE

```
IF 30>20 AND 20<30 THEN PRINT "TRUE "
IF "Hi"="hello" AND 6-5=1 THEN PRINT "TRUE TOO!"

RUN

TRUE
```

```
PRINT BIN$( &X00001111 AND &X11111111)
PRINT 4 AND 255

RUN

0000000000001111
4
```

REMARK See OR, XOR and NOT.

AND TRUTH TABLE

condition AND condition TRUE(-1) if both conditions TRUE, else FALSE(0)

AND	
1 AND 1	= 1
0 AND 1	= 0
1 AND 0	= 0
0 AND 0	= 0

BOOLEAN "16 BIT" LOGIC		
00000001		00000111
AND 00001111	AND	00001111
= 00000001	=	00000111

LongInteger will function with this operator in 32 bits.

command **APPEND**

FORMAT

APPEND line or label ["] *filename* ["]
APPEND* line or label ["] *filename* ["]

DEFINITION

Used to append or insert a program segment or subroutine (saved with SAVE+) into the present program in memory.

A non-line numbered ASCII program file is required to append a subroutine into the present program in memory at the specified line number . Line numbers will be assigned in increments of one.

APPEND* will strip REM(arks) and spaces to free up more memory for the program as the program in inserted.

EXAMPLE

```
10 "TEST ROUTINE"
20 FOR I = 1 TO 10
30   PRINT I
40 NEXT I
50 RETURN

SAVE+ TEST.APP

APPEND 31 TEST.APP

LIST

00010 "TEST ROUTINE"
00020 FOR I = 1 TO 10
00030   PRINT I
00031   "TEST ROUTINE"     <----Subroutine inserted here
00032   FOR I = 1 TO 10    <---- (Example only, program will not run)
00033     PRINT I
00034   NEXT I
00035   RETURN
00040 NEXT I
00050 RETURN
```

REMARK

The program to be appended must be in ASCII format and not contain line numbers. Use the SAVE+ command to save programs without line numbers.

If any line number being used in APPEND already exists, it will overwrite the existing line. Also see MERGE, LOAD, SAVE, SAVE*, SAVE+.

ASC function

FORMAT **ASC** (*string*)

DEFINITION Returns the ASCII code value (a number between 0 and 255) of the first character in a string. ASCII stands for American Standard Code for Information Interchange.

EXAMPLE

```
PRINT ASC("A"), ASC("B")
PRINT CHR$(65), CHR$(66)
PRINT ASC("America")

RUN

65              66
A               B
65
```

REMARK ASC returns 0 if the length of string is zero or the ASCII code of the string is zero. Use this logic to determine the true status if an ASCII zero is the result:

```
LONG IF ASC(A$)=0 AND LEN(A$)>0
  PRINT "ASCII code of A$ =0"
XELSE
  PRINT"A$ is an empty string"
END IF
```

The inverse function of ASC is CHR$. To return the character represented by the ASCII code, use CHR$(*ASCII number*).

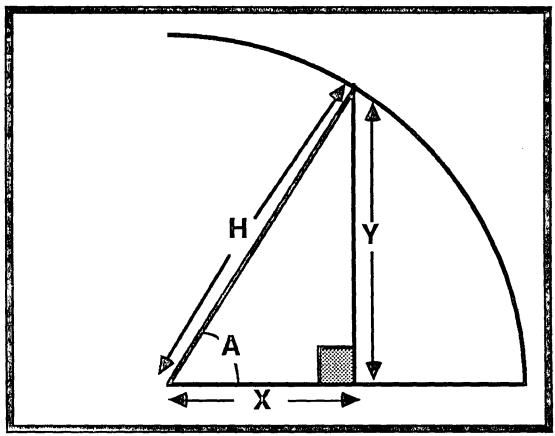
ASCII codes may vary from machine to machine.

ASCII codes 32 through 127 are usually the same for all microcomputers. See CHR$ with example ASCII listing.

function **ATN**

FORMAT **ATN** (*expression*)

DEFINITION Returns the angle, in radians, for the inverse tangent of *expression*.



A=**ATN**(Y/X), PI=**ATN**(1) << 2

EXAMPLE

```
Pi#=ATN(1) << 2
PRINT Pi#

RUN

3.141592... <---Based on digits of accuracy set in configuration.
```

REMARK ATN is a scientific function. Using ATN in an expression will force ZBasic to calculate that part of an expression in Double Precision.

ZBasic allows you to configure the accuracy for scientific functions separately for both Double and Single Precision. See "Configure".

Also see "Expressions" and "Derived math functions" in the "MATH" section of this manual.

AUTO command

FORMAT

AUTO
AUTO *starting line*
AUTO *starting line*, *increment*
AUTO , *increment*

DEFINITION

This command automatically generates line numbers in the Standard Line editor to save time. The two optional parameters are:

starting line	Starting line number	(default is 10)
increment	Line spacing	(default is 10)

To end AUTO line numbering press either <BREAK> or <CTRL C> at the first line number you will not use.

EXAMPLE

```
AUTO

10                    <---- Type in text then <ENTER> to go to next line.
20
30 <BREAK>

AUTO   100,20

100
*120                  <---- Careful, this line already exists!!
130 <BREAK>
```

REMARK

An asterisk appearing before a line number indicates an occupied line. Pressing <ENTER> will skip that line leaving the original contents intact and resume auto line numbering with the next line. To remove the line type a space and <ENTER>.

Also see LIST, EDIT

statement **BEEP**

FORMAT **BEEP**

DEFINITION Sounds the speaker.

EXAMPLE

```
FOR X=1 TO 10
  BEEP
NEXT

RUN

BEEP, BEEP...
```

REMARK Also see SOUND.

BEEP is not supported with Apple // or Z80 computers. For Apple // and most CP/M computers use `PRINT CHR$(7)` instead. See your SOUND and your computer appendix for other ways of creating audio output.

BASE OPTION configuration

FORMAT **Array Base 0 or 1?**

DEFINITION An option in the ZBasic configuration routine to set the array BASE to either zero or 1. The default is zero.

EXAMPLE See "Configure" in the beginning of this manual for an explanation of configuring your version of ZBasic to your preferences.

ARRAY BASE ZERO

```
DIM A(100)          <-- elements 0-100 (101 elements)
DIM Tables(22)      <-- elements 0-22 (23 elements)
```

ARRAY BASE ONE

```
DIM A(100)          <-- elements 1-100 (100 elements)
DIM Tables (22)     <-- elements 1-22 (22 elements)
```

REMARK See DIM and "Array Variables".

function **BIN$**

FORMAT **BIN$** (*expression*)

DEFINITION Returns a 16 character string which represents the binary (BASE 2) value of the result of the integer *expression*. Some typical binary numbers:

```
0000000000000001    =   1
0000000000000011    =   3
0000000000000111    =   7
0000000011111111    =   255
0000000100000000    =   256
1111111111111111    =   -1 (65,535 unsigned)
```

EXAMPLE The following program will convert a decimal number to binary or a binary number to decimal:

```
"Binary Conversion"
CLS
DO
  INPUT"Decimal number to convert: ";Decimal%
  PRINT BIN$(Decimal%)
  INPUT"Binary number to convert: ";Binary$
  Binary$="&X"+Binary$
  PRINT VAL(Binary$)
UNTIL Decimal% = 0
```

```
RUN

Decimal number to convert: 255
0000000011111111

Binary number to convert: 0000000000000011
3
```

REMARK Note that conversions are possible from any base to any other base that ZBasic supports. &X is the inverse function of BIN$.

Also see HEX$, OCT$, UNS$ and "Numeric Conversions".

Use DEFSTR LONG to set BIN$ and &X to work in LongInteger (32bits).

BOX statement

FORMAT

BOX [TO] *expr*$_{x1}$, *expr*$_{y1}$ [**TO** *expr*$_{x2}$, *expr*$_{y2}$ ...]
BOX FILL [TO] *expr*$_{x1}$, *expr*$_{y1}$ [**TO** *expr*$_{x2}$, *expr*$_{y2}$ ...]

DEFINITION

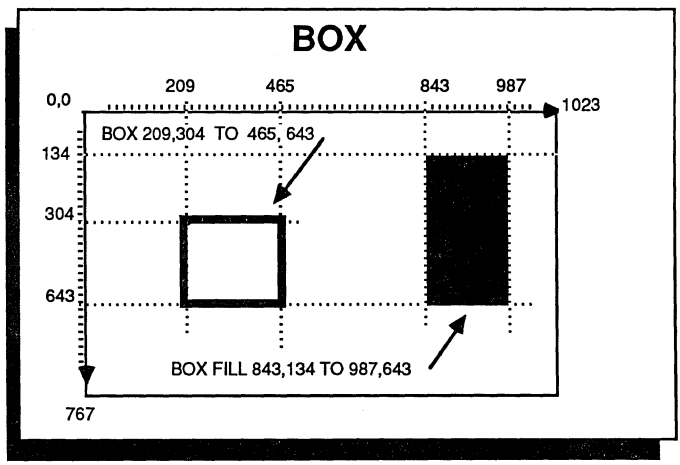
Draws a BOX from the coordinates defined by the first corner (x1,y1) to the coordinates defined by the opposite corner (x2,y2) in the current COLOR.

If BOX TO x,y is used the first corner will be the last graphic point used. If undefined then 0,0 will be the default.

If the optional FILL appears directly after the command, the BOX will be painted as a solid BOX in the current color.

The default screen positions are given using Device Independent Coordinates of 1024 across by 768 down.

EXAMPLE



REMARK

The output will vary depending on the graphic capability of the host computer. Also see CIRCLE, MODE, FILL, PLOT, RATIO and COLOR.

statement **CALL**

FORMAT

CALL *number*
CALL LINE *line or label*

DEFINITION

CALL will execute a machine language subroutine at the address specified by *number* or the address of the compiled *line* .

EXAMPLE

Use these examples only if you understand machine language.

```
REM TRS80 I & III, CALL DEBUG
CALL &H440D
:
REM CPM 80, CALL WARM START (Exits to DOS)
CALL 0
:
REM APPLE CALL TO SOUND BELL TONE
CALL  -198
:
10 REM CALL LINE examples
20 CALL LINE 40
30 CALL LINE  "LABEL"
40 MACHLG 34, 21, x%, 255, 9: RETURN
50 "LABEL":  MACHLG . . .  :  RETURN
```

REMARK

CALL is useful for transferring program control to a machine language subroutine from which a return to the ZBasic program is desired. The routine to be called must be terminated by that machine's instruction for RETURN.

Also see MACHLG, USR, LINE and DEFUSR.

WARNING*:* Use of this command requires an understanding of machine language programming and the computer hardware being used. Porting of this code may not be possible without re-writing the machine language routines.

See CALL in your appendix for enhancements.

CASE statement

FORMAT

```
SELECT [CASE] [expression]
    CASE [IS] relational condition [,  relational condition] [,...]
        statement [:statement:...] ]
    CASE [IS] condition [, condition] [,...]
        statement [:statement:...] ]
    CASE boolean expression
        statement [:statement:...] ]
    CASE  ELSE
        statement [:statement:...] ]
END SELECT
```

DEFINITION

When SELECT/CASE is encountered, the program checks the value of the controlling expression or variable, finds the CASE that compares true and executes the statements directly following the CASE statement. After these statements are performed, the program continues at the line after the END SELECT statement:

CASE relational,...

If the expression after SELECT compares true to any one of a number of relational conditions, the statements following the CASE are executed and the program continues after the END SELECT:

```
SELECT 12
  CASE >10
    PRINT "This is the right answer"
  CASE >20, <10
    PRINT "This is not true"
END SELECT
program continues here...
```

CASE *condition,...*

If the expression following the SELECT equals any one of a number of *conditions* the statements following the CASE are executed (program continues after the END SELECT).

```
A=23
SELECT A
  CASE 10
    PRINT "This is the wrong answer"
  CASE 10,23,11,10
    PRINT "This would be true"
END SELECT
```

CASE boolean

If an expression after SELECT is omitted, you may use a boolean or TRUE/FALSE condition. The statements after the first TRUE (non-zero) CASE condition will be executed. Only one boolean statement is allowed following CASE.

```
A=10:B=20
SELECT
  CASE (A=10 AND A>20)
    PRINT "This is the correct answer"
  CASE (A>B OR A=B)
    PRINT "This is the wrong answer"
END SELECT
```

CASE ELSE — If all of the CASE statements in the SELECT CASE structure are false the statements following the CASE ELSE are executed.

```
"Start"
A$="Maybe"
SELECT A$
  CASE "Yes"
    PRINT "Thank you for saying Yes"
  CASE "No"
    PRINT "Thank you for saying No"
  CASE ELSE
    PRINT "You smart aleck!"<--Does this one
END SELECT
```

REMARK This is a powerful structured way of doing complicated IF-THEN-ELSE or LONG IF statements especially when there are multiple lines of complicated comparisons.

This structure is also much easier to read than complicated IF statements.

See SELECT for more information.

Important Note: Never exit a SELECT CASE structure using GOTO. This will introduce problems into the stack and cause unpredictable system errors. Always exit the structure at the END SELECT. Be sure to enclose loops and other contructs completely within the SELECT-CASE and CASE ELSE constructs.

The Z80 versions do not support SELECT CASE. See LONG IF and IF for ways of doing the same thing.

The Apple DOS 3.3 and ProDOS versions does not support SELECT CASE. See LONG IF and IF for ways of doing the same thing.

CHR$ function

FORMAT **CHR$** (*expression*)

DEFINITION Returns a single character string with the ASCII value of the result of *expression*. The range for the value of *expression* is 0 to 255.

The inverse function of CHR$ is ASC;

EXAMPLE

```
"Print ASCII character set for this computer"
CLS
REM  Use ROUTE 128 here to send output to printer.
FOR I=32 TO 127 STEP  8
  FOR J= 0 TO 7: X =I+J
    PRINT USING "###=";X;CHR$(X);"  ";
  NEXT J  :PRINT
NEXT I
```

RUN

```
 32=     33=!    34="    35=#    36=$    37=%    38=&    39='
 40=(    41=)    42=*    43=+    44=,    45=-    46=.    47=/
 48=0    49=1    50=2    51=3    52=4    53=5    54=6    55=7
 56=8    57=9    58=:    59=;    60=<    61==    62=>    63=?
 64=@    65=A    66=B    67=C    68=D    69=E    70=F    71=G
 72=H    73=I    74=J    75=K    76=L    77-M    78=N    79=O
 80=P    81=Q    82=R    83=S    84=T    85=U    86=V    87=W
 88=X    89=Y    90=Z    91=[    92=\    93=]    94=^    95=_
 96=`    97=a    98=b    99=c   100=d   101=e   102=f   103=g
104=h   105=i   106=j   107=k   108=l   109=m   110=n   111=o
112=p   113=q   114=r   115=s   116=t   117=u   118=v   119=w
120=x   121=y   122=z   123={   124=|   125=}   126=~   127=#
```

```
PRINT CHR$(64)
PRINT ASC("A")
```

RUN

```
A
64
```

REMARK When the program above is run, the character set for that computer will be displayed. Some of the characters above may differ from what you get on your system. Try changing the range above from 127 to 255. Some computers have extra characters or graphic symbols for these codes.

Characters in the range of 0-31 are usually reserved for control codes like linefeed (10), carriage return (13)...

If the PRINT statement is changed to LPRINT the printer's character set will be printed. If *expression* is less than 0 or greater than 255, only the low order byte will be used.

```
CHR$(256)  =  CHR$(0)
CHR$(257)  =  CHR$(1)
```

statement **CIRCLE**

FORMAT

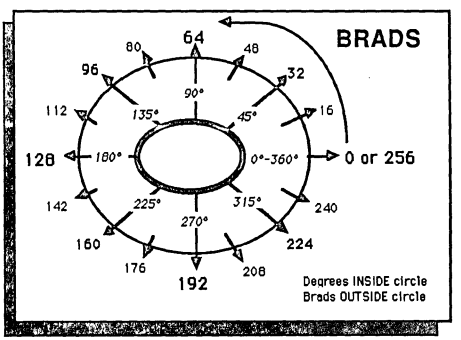
CIRCLE [FILL]	*expr*$_1$,	*expr*$_2$,	*expr*$_R$			
CIRCLE	*expr*$_1$,	*expr*$_2$,	*expr*$_R$	**TO**	*expr*$_S$,	*expr*$_B$
CIRCLE	*expr*$_1$,	*expr*$_2$,	*expr*$_R$	**PLOT**	*expr*$_S$,	*expr*$_B$

DEFINITION

Draws a CIRCLE in the current COLOR.

If the optional FILL is used directly after the command, the CIRCLE will be filled with the current COLOR. If TO is used, a PIE segment will be displayed (shaped like pie slices). If PLOT is used, only the ARC segment will be displayed (a segment of the circumference).

expr$_1$	horizontal center
expr$_2$	vertical center
expr$_R$	radius (diameter of circle) in graphic coordinates
expr$_S$	start of angle in brads (zero starts at 3:00 o'clock)
expr$_B$	Number of brads to draw ARC or PIE (counter clockwise).



EXAMPLE

SEE ILLUSTRATIONS ON FOLLOWING PAGE.

REMARK

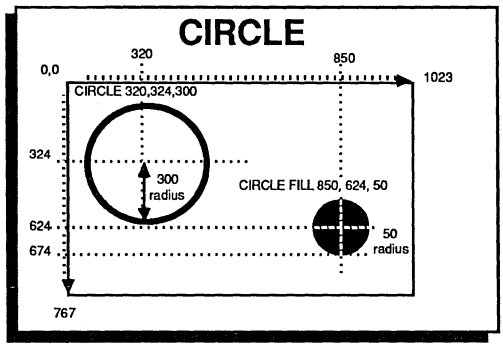
CIRCLE uses the ZBasic Device Independent Graphic Coordinates of 1024 x 768. For more details see the CIRCLE in the "Graphics" section in this manual. Also see RATIO, MODE, PLOT, COLOR, FILL and BOX.

Macintosh: See COORDINATE WINDOW for pixel coordinates and toolbox for ways of using QuickDraw for creating boxes. **MSDOS**: See COORDINATE WINDOW for converting to pixel coordinates. **Apple**: See appendix for ways of converting to pixel graphics.

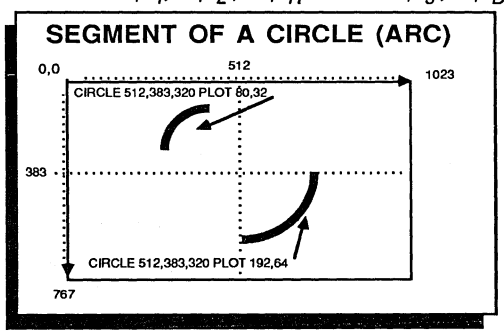
CIRCLE statement

EXAMPLE

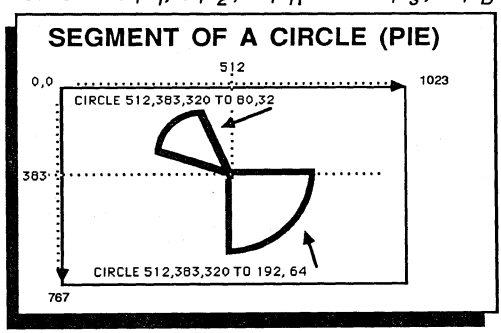
CIRCLE $expr_1$, $expr_2$, $expr_R$
CIRCLE FILL $expr_1$, $expr_2$, $expr_R$



CIRCLE $expr_1$, $expr_2$, $expr_R$ **PLOT** $expr_S$, $expr_B$



CIRCLE $expr_1$, $expr_2$, $expr_R$ **TO** $expr_S$, $expr_B$



statement **CLEAR**

FORMAT

CLEAR
CLEAR *number*
CLEAR END
CLEAR INDEX$

DEFINITION

Used to reserve memory or clear all or specified variables (sets the values of the variables to null or zero).

CLEAR	Sets all variables and INDEX$ to zero or null.
CLEAR *number*	Sets aside *number* bytes for the INDEX$ array.
CLEAR END	CLEARS all variables which have not yet been assigned in the program.This form of CLEAR is normally used to clear all variables not being used when chaining. See "Chain" in the front section for more information.
CLEAR INDEX$	Sets all elements of the INDEX$ array to null.

EXAMPLE

```
INPUT"Name: ";Name$
PRINT Name$
CLEAR
PRINT Name$

RUN

Fred
      <-----Nothing printed here since Name$ was cleared  at line 3.
```

REMARK

Only one CLEAR *number* is allowed in a program and must appear before any variables are encountered. Be sure to CLEAR one extra byte for each element in the INDEX$ array. Also see "Special INDEX$ Array" and "CHAIN".

A CLEAR is performed at the beginning of each program created with RUN or RUN*. RUN+ or warm start programs will not CLEAR variables at startup.

See INDEX$ in Mac appendix for added enhancements available on this version.

CLOSE statement

FORMAT **CLOSE** [[#] *expression*$_1$ [, [#] *expression*$_2$, . . .]]

DEFINITION This statement is used to CLOSE one or more OPEN files or other devices.

The parameter *expression* indicates a device number or file number.

If no file or device numbers are declared all OPEN devices will be closed.

EXAMPLE

```
OPEN"I",1,"FILE1",10
OPEN"I",2,"FILE2",100
READ#1, A$;10
READ#2, B$;10
CLOSE#1,2                  <---File1 and 2 are closed
OPEN"R",1,"FILE3"          <---File1 may now be used again
CLOSE                      <----All files are closed
```

REMARK All files should be closed before leaving a program to insure that data will not be lost or destroyed. If program exit is through END or STOP, all files will be closed.

statement **CLS**

FORMAT

CLS
CLS *expression*
CLS LINE
CLS PAGE

DEFINITION

These statements will clear all, or portions, of the screen of text and graphics.

CLS	Clears the entire screen of text and graphics. Cursor ends up at the top left corner of screen.
CLS *expression*	In TEXT mode this fills screen with the ASCII character specified by*expression* and places the cursor at the top left corner of the screen*.
CLS *expression*	In GRAPHICS mode this will fill the screen with the color specified by *expression.*
CLS LINE	Clears from the cursor position to the end of the line. Cursor will remain where it was.
CLS PAGE	Clears from the cursor position to the end of the screen. Cursor will remain where it was.

EXAMPLE

```
CLS
CLS 65          <----Fills screen with A's
CLS ASC("*")    <----Fills screen with *'s
LOCATE 0,10
CLS LINE        <----Clears line 10 of text and graphics
LOCATE 0,12
CLS PAGE        <----Clears screen from line 12 down.
```

REMARK

See LOCATE, PRINT@, PRINT%, FILL and MODE. See your computer appendix for possible variations.

CLS clears the current window (not the entire screen). CLS *expression* will clear the screen with white if *expression*=0 and black if *expression* ><0.

COLOR statement

FORMAT **COLOR** [=] *expression*

DEFINITION Sets the **COLOR** to be used by all graphic drawing commands. Color values will vary from one computer to the next. See your computer appendix for specifics. For most computers 0 is the background color and -1 is the foreground color.

If you have a black and white monitor, 0 is Black, -1 is white.

If your computer is incapable of graphics or you are using one of the character modes, the expression will determine the ASCII character to be used. (With some graphics modes, zero = space , all others = asterisk "*").

EXAMPLE

```
CLS: MODE  6          <----even modes are character graphics with some versions
COLOR ASC("*")        <----Uses asterisks for graphics (not all versions)
PLOT 0, 256
MODE=7                <---- odd modes are actual graphics
CIRCLE 768,200,50
COLOR=6               <----Sets COLOR to 6
BOX 0,0 TO 10,10
END
```

REMARK Also see MODE, PLOT, CIRCLE, BOX, POINT and FILL. Colors vary by mode, graphic type, monitors and other hardware criteria. Check hardware manual and the ZBasic appendix for your computer for specific color codes.

Macintosh: NOT(0) =black, 0=white. See appendix for variations especially with Macintosh II which supports a number of colors and grey levels.

MSDOS: COLOR is also used to change text color, background color, blinking, underline etc. See appendix for specifics. See CGA colors below.

Apple: Color chart below and the Apple appendix.

TRS-80 and **Kaypro**: Black=0, -1=white.

EXAMPLE COLORS CODES

IBM PC and compatibles

CGA MODE 5	
0= BLACK	8= GRAY
1= BLUE	9= LT BLUE
2= GREEN	10= LT GREEN
3= CYAN	11= LT CYAN
4= RED	12= LT RED
5= MAGENTA	13= LT MAGENTA
6= BROWN	14= YELLOW
7= WHITE	15= Bright WHITE

Apple // ProDOS and **DOS 3.3**

MODE 5	MODES 1,3 and 7	
0=BLACK1	0=BLACK	8=BROWN
1=GREEN	1=MAGENTA	9=ORANGE
2=VIOLET	2=DARK BLUE	10=GREY
3=WHITE1	3=PURPLE	11=PINK
4=BLACK2	4=DARK GREEN	12=GREEN
5=ORANGE	5=GREY	13=YELLOW
6=BLUE	6=MED. BLUE	14=AQUA
7=WHITE2	7=LIGHT BLUE	15=WHITE

statement **COMMON**

FORMAT COMMON variable list...

DEFINITION Identical to the ZBasic DIM statement. It is used to allocate memory for variables and for declaring variables common to chained programs.

The order of the variables declared in COMMON is important when chaining programs. The COMMON statement in one program must be exactly the same and in exactly the same order in other programs being chained.

EXAMPLE See DIM.

REMARK See DIM and "Chaining" in this manual.

This statement is added to make ZBasic compatible with other versions of BASIC.

Not available on the Apple // or Z80 versions of ZBasic. Use DIM.

COMPILE command

FORMAT **[L] COMPILE**

DEFINITION Compiles a program and lists all of the compile time errors that are encountered.

If optional "L" is used, the error listings are sent to the printer.

This command is essentially the same as RUN except the compiler does not stop at the first error.

EXAMPLE

```
PWINT "Hello"
X=X+1
INPUT "Yes or No:"A$
GOSUB "Routine"
END

COMPILE

Syntax Error in Stmt 01 at Line 00001
00001 PWINT "Hello"

";" Expected Error in Stmt 01 at line 00003
00003 INPUT "Yes or No:"_A$

Line# Error in Stmt 01 at Line 00004
00004 GOSUB "Routine"
```

REMARK See RUN and the section in the front of the manual called "Errors".

Not supported. Use RUN.

Not supported. Use RUN.

command **CONFIG**

FORMAT **CONFIG**

DEFINITION Invokes the configuration prompts that allow you to set preferences for a number of items including:

Digits of precision
Default variable types
Integer or floating point expression evaluation
Spaces between keywords
Convert to uppercase
Number of files that can be opened
The Rounding factor for PRINT USING
Test Array bounds

and a number of special options for your computer.

EXAMPLE See "Configure" in the front of this manual and the section in your appendix for specific configuration options available for your version of ZBasic.

REMARK This command is not available on all versions. See below.

The Z80 versions of ZBasic do not offer this command. The option to configure is offered only when you first load ZBasic.

CONFIG is not offered as a command but "Configure" is always available as a menu item. See appendix for the options specific to this version.

COORDINATE statement

FORMAT **COORDINATE** [[**WINDOW**] *horizontal, vertical*]

DEFINITION Allows you to change the coordinate system used for graphic functions and statements.

ZBasic defaults to a coordinate system of 1024 x 768. This allows programs created on one computer work on other computers with different graphic hardware.

COORDINATE *horiz, vert*	Sets the relative coordinate system to the specified limits minus one. COORDINATE 100, 100 would allow setting the coordinates from 0 to 99 for both the horizontal and vertical.
COORDINATE WINDOW	Sets the system to pixel coordinates. This allows you calculate the graphic positions by the actual resolution on the screen. While this is not recommended for programs that will be ported to other computers, some people prefer it for certain applications.

EXAMPLE

```
PLOT 1023, 767              <--- Puts a graphic dot at the ZBasic
:                                default coordinates (lower right corner)
COORDINATE WINDOW
PLOT 100,100                <--- Puts a graphic dot at the pixel coordinate
:
COORDINATE 1000,500
PLOT 100,100                <--- Puts a graphic dot at the relative coordinate
```

REMARK Some versions do not support this statement. See below for alternatives to changing coordinate systems.

Not supported on Z80 versions although COORDINATE WINDOW may be emulated by using this instruction: `POKE&xx3F,&C9` to enable pixel graphics and `POKE&xx3F,&C3` to return to the default coordinates of 1024x768. The value of xx varies by version type: CP/M-80=01, TRS-80 1,3=52 and TRS-80 model 4=30.

Not supported on these versions although COORDINATE WINDOW may be emulated using the statements below:

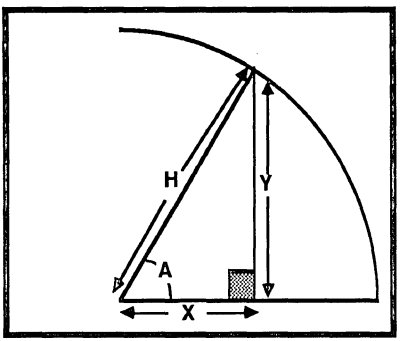
Apple ProDOS: `POKEWORD &85,0` for pixel coordinates for that mode of graphics. Use `MODE` to set back to regular coordinates.

Apple DOS 3.3: `POKE &F388,&60` for pixel coordinates of that mode. `POKE &F388,&A9` to set back to the default coordinates of 1024x768.

function **COS**

FORMAT **COS** (expression)

DEFINITION Returns the Cosine of the *expression* in radians.



COS(A)=X/H, H***COS**(A)=X, X/**COS**(A)=H

EXAMPLE

```
X#=COS(X)
```

REMARK Using COS in an expression will force ZBasic to calculate that expression in floating point. COS is a scientific function. You may configure BCD scientific accuracy separately from both Double and Single Precision immediately after loading ZBasic.

Integer Cosine may be accomplished with the predefined ZBasic USR function; USR9 (angle in Brads). This returns the integer cosine of an angle in the range ±255 (corresponding to ± 1). The angle must be in Brads. This example program will draw a sine wave using USR9:

```
MODE7 :CLS
FOR I=0 TO 255
  PLOT I<<2,-USR9(I)+384
NEXT I
```

For more information about scientific functions and derived math functions see the "Math" section of this manual. See CIRCLE for more about BRADS. Also see ATN, SIN, TAN, EXP, SQR.

CSRLIN function

FORMAT **CSRLIN**

DEFINITION Returns the line where the cursor is positioned.

EXAMPLE

```
CLS
PRINT
PRINT
PRINT CSRLIN

RUN

2
```

REMARK See POS to determine the horizontal cursor position.

Not supported with the Apple // or Z80 versions of ZBasic. For Apple // use `PEEK(37)` to get the current cursor line.

function **CVB**

FORMAT **CVB** (*string*)

DEFINITION Returns the binary floating point value of the first *n* characters of the condensed number in *string* (depending on whether Single or Double Precision is used).

Double Precision	Returns the digits of accuracy defined in configure for double precision. (default is 8 digits i.e. the first 8 string characters.)*
Single Precision	Returns the digits of accuracy defined in configure for single precsion. (default is 4 digits i.e. the first 4 string characters.)

This function is the compliment of MKB$.

EXAMPLE

```
A#=12345.678: B!=12345.678
:
A$=MKB$(A#): B$=MKB$(B!)
PRINT LEN(A$), LEN(B$)
:
C#=CVB(A$): D!=CVB(B$)
PRINT C#, D!

RUN

8             8
12345.678     12345.7
```

REMARK This function is used with some versions of BASIC to save space on disk when storing large amounts of numeric data in strings with FIELD. ZBasic does this automatically but CVB is still useful for string packing, etc. Also see MKI$, CVI, MKB$, READ# AND WRITE#. This command is not compatible with CVS or CVD.

A few things to remember concerning CVB:

Null strings or 1 character strings return 0
Two character strings will return 2 digits accuracy. Four character strings will return four digits. See "Floating Point Variables" for more information.

*See "Floating Point Variables" for detailed information on how extended double precision variables are stored and the added range of this precision for the Mac.

CVI function

FORMAT **CVI** (*string*)

DEFINITION Returns the binary integer value of the first 2 characters of *string*.

This function is the compliment of MKI$.

EXAMPLE

```
A$=MKI$(30000)
PRINT LEN(A$)
:
Z%=CVI(A$)
PRINT Z%
END

RUN

 2
 30000
```

REMARK Also see MKI$, CVB, MKB$, READ# AND WRITE#.

A few things to remember concerning CVI:

Null string returns 0
One character strings will return the ASCII value.
Two character strings will return an integer value.
ASC(second character) * 256 + ASC(first character)

This function was used with MBASIC to save space on disk when storing large amounts of numeric data. ZBasic does this automatically when using WRITE# and READ# but CVI is still useful for string packing, etc.

See DEFSTR LONG in the Mac appendix for using this function with LongIntegers. When LongIntegers are used the memory requirements are four bytes instead of two bytes. MSB and LSB are stored in reverse order for regular integers with this version.

statement **DATA**

FORMAT **DATA** *data item* [, *data item* [, . . .]]

DEFINITION The DATA statement is used to hold information that may be read into variables using the READ statement. DATA items are a list of string or numeric constants separated by commas and may appear anywhere in a program.

No other statements may follow the DATA statement on the same line.

Items are read in the order they appear in a program. RESTORE will set the pointer back to the beginning of the first DATA statement. RESTORE *n* will set the pointer to the *n*th DATA item.

EXAMPLE

```
DATA Tom, Dick, Harry, 12.32, 233
READ A$, B$, C$, A#, B%
:
DEF TAB 6
PRINT "DATA items are: ";A$,B$,C$,A#,C%

RUN

 DATA items are: Tom    Dick     Harry     12.32     233
```

```
DATA Tom, Dick, Harry, 12.32, 233
:
RESTORE 3
READ Name$
:
PRINT "Third DATA item is: ";Name$

RUN

Third DATA item is: Harry
```

REMARK Alphanumeric string information in a DATA statement need not be enclosed in quotes if the first character is not a number, math sign or decimal point.

Leading spaces will be ignored (unless in quotes). DATA statements can be included anywhere within a program and will be read in order.

Typical storage requirements for DATA items:

Number with zero value	2 bytes
Non-zero integer	3 bytes
Strings	Length of string + 2
Floating Point BCD	"See Floating Point Constants"
Floating Point Binary	"See Floating Point Constants"

See READ, PSTR$ DIM and RESTORE for common statements used with DATA.

Note: See PSTR$ for extremely efficient way of retrieving strings in DATA statements.

DATE$ function

FORMAT **DATE$**

DEFINITION Returns an eight character string containing the system date using the format MM/DD/YY, where MM=month, DD=day and YY=year.

EXAMPLE

```
DATA January, February, March, April, May, June
DATA July, August, September, October, November, December
:
A$=DATE$
:
Day$=MID$(A$,4,2)
REM  If leading zero; peel off on next line
IF ASC(Day$)=ASC("0") THEN DAY$=RIGHT$(DAY$,1)
:
Month%=VAL(A$)
RESTORE Month%
READ Month$                <---Get month name from DATA
:
Year$="19"+RIGHT$(A$,2)
:
PRINT "Computer date: ";TAB(20);DATE$
PRINT "Human date: ";TAB(20);Month$;" ";Day$;", ";Year$

RUN

Computer date:   08/03/88
Human date:      August 3, 1988
```

REMARK If the system does not support a date function, 00/00/00 will be returned. See your computer appendix for more information.

Also see TIME$ and DELAY.

Macintosh: Date can only be changed from the "Control Panel DA"

MSDOS: Date may be set in program: DATE$="MM/DD/YY"

Apple: Date must be set from the system.

CP/M-80 3.0 and **PLUS**: DATE$ supported. CP/M 2.x does not support date.

statement **DEF**

FORMAT

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

DEFINITION

These statements define which variable type ZBasic will assume when encountering a variable name with *letter* as a first character and not followed by a type declaration symbol (% integer, ! single, # double, $ string, & double integer).

DEFINT	Integer
DEFSNG	Single Precision
DEFDBL	Double Precision
DEFSTR	String
*DEFDBL INT	LongInteger (Macintosh only)

ZBasic will assume that all variables are integers unless followed by a type declaration symbol or defined by a DEF *type* statement.

See "Configure" for another way of defining the default variable type.

letter	Letter from A to Z. Case is not significant.
letter - letter	Defines an inclusive range of letters.

EXAMPLE

```
DEFSNG A              <--- A and A! are the same variable (A$ is still a string).
DEFDBL B              <--- B and B# are the same variable (B% is still an integer).
DEFINT F              <--- F and F% are the same variable (F! is still single prec).
DEFSTR Z              <--- Z and Z$ are the same variable (Z# is still double prec).
DEFSTR B-D, X,Y,Z     <--- B, C, D, X, Y and Z all strings
DEFDBL A, F-J, T      <--- A, F,G,H,I,J and T all Double precision
DEFSGL A, G, B-E      <--- A, G, B, C, D and E all Single Precision
```

REMARK

Other versions of BASIC may assume all numeric variables are single precision unless otherwise defined. See the sections on "Floating Point Variables", "Math" and "Converting Old Programs" in the front of this manual for more information.

*Also see DEFSTR LONG in appendix for way of forcing HEX$, OCT$, UNS$, CVI and MKI$ to default to LongInteger instead of regular integer.

DEF FN statement

FORMAT **DEF FN** *name* [(*variable* [, *variable* [, ...]])] = *expression*

DEFINITION This statement allows the user to define a function that can thereafter be called by FN *name*. This is a handy way of adding functions not provided in the language.

The *expression* may be a numeric or string expression and must match the type the FN *name* would assume if it was a variable *name*.

The *name* must adhere to variable name syntax.

The *variable* used in the definition of the function is a dummy variable. When using FN the dummy variables, other variables or expressions may be used to pass the values to the function. The *variable* should be of the right type used in the function.

EXAMPLE

```
DEF FN e# = EXP(1.)
DEF FN Pi#= ATN(1)<<2
DEF FN Sec#(x#) = 1.\COS(x#)
DEF FN ArcSin#(x#) = ATN (x# \ SQR( 1 - x# * x#))
:
PRINT FN Pi#
I#=4.2312
Planet#= FN ArcSin#(Sin(I#))* FN e#+ FN Sec#(Elipse#)

RUN

3.14159...
```

```
REM     A Handy rounding function
REM     Send the routine the number and places to round
:
DEF FN Round#(num#, places)=INT(num#*10^places+.5)/10^places
:
PRINT FN Round#(823192.124567576,5)
X#=202031.12332
PRINT FN Round#(X#,2)
END

RUN

823192.12457
202031.12
```

REMARK One function may call another function as long as the function was defined first.

LONG FN is another form of DEF FN that allows multiple lines of code. It is very powerful for creating reusable subroutines.

See "Derived Math functions", "Functions and Subroutines", LONG FN, END FN and FN.

statement **DEF LEN**

FORMAT **DEF LEN** [=] *number*

DEFINITION The DEF LEN statement is used to reset the default length of string variables until the next DEF LEN statement is encountered. The *number* must be from 1 to 255.

If DEF LEN is not used string length default is 255 characters each. Each string will consume 256 bytes; 1 byte for length byte, the rest for characters.

Since strings will consume so much memory if their length is not defined; it is imperative that thought be given to string length, especially if memory is at a premium.

EXAMPLE

```
C$="Welcome"            <---Length of C$ defaults to 255 characters.
:
DEF LEN 20
DIM A$(10)              <---A$() allocated 20 characters per element.
Greeting$="Hello"       <---Greeting$ allocated 20 characters
:
DEF LEN 200
B$="Goodbye"            <---B$ allocated 200 characters
:
DIM 50 Z$               <---Z$ allocated 50 characters. See DIM
```

REMARK DEF LEN will allocate the specified amount of memory to every string that is defined after it (unless defined differently in DIM or another DEF LEN).

Strings that appear before the DEF LEN statement are not affected. For example, in the above program, C$ is allocated the default length of 255 characters because it appeared BEFORE the DEF LEN statement.

DIM may also be used to set the length of string variables. See DIM.

Also see "String Variables" and "Converting Old Programs" in the front section for important information about strings and how they use memory.

Important Note: Always allocate one extra character for strings used with INPUT. Never use a one character string for INPUT. The extra character position is needed for the carriage return.

DEF MOUSE statement

FORMAT **DEF MOUSE** [=] *expression*

DEFINITION The DEF MOUSE statement is used to define the device to be used with the MOUSE functions and statements, or the type of mouse commands to use with the program.

DEF MOUSE=0 Regular ZBasic MOUSE commands for a mouse device. See MOUSE in this reference section.

MSDOS: Uses Microsoft™ compatible mouse devices. Be sure to "Configure" ZBasic for a mouse.

Apple //: Assumes a mouse is connected.

Macintosh: Standard MOUSE commands in this section of the reference manual. See DEF MOUSE=1 to do MSBASIC type mouse commands.

Z80: NOT SUPPORTED.

DEF MOUSE= n Tells ZBasic that other devices are to be used instead of a MOUSE (in the case of the Macintosh it tells ZBasic to use MSBASIC mouse syntax).

MSDOS: n=1 defines joystick/paddle A*
n=2 defines joystick/paddle B*
n=3 defines a lightpen device

Apple //: n=1 defines a joystick/paddle device*

*MOUSE(3) function returns button status:
0= No button pressed
1= Button zero pressed
2= Button one pressed
3= Both buttons pressed

Macintosh: n= non-zero sets commands to MSBASIC mouse commands. See Macintosh appendix for specifics.

Z80: NOT SUPPORTED.

EXAMPLE See the appendix for your computer for specifics.

REMARK See MOUSE in this reference section and in your appendix for specifics.

MOUSE or DEF MOUSE is not supported with any Z80 versions of ZBasic. This is due to the fact that most Z80 computers do not offer this hardware device.

statement **DEF TAB**

FORMAT **DEF TAB** [=] *expression*

DEFINITION The DEF TAB statement is used to define the number of characters between tab stops for use in PRINT, PRINT# or LPRINT statements

Tab stops are the number of spaces to move over when the comma is encountered in a PRINT statement.

The *expression* must be a number from 1 to 255. TAB default is 16.

EXAMPLE

```
PRINT 1,2,3     <---Tab stop default is 16, 32, 48...
DEF TAB = 8     <---Tab stops now set to 8, 16, 24...
PRINT 1,2,3: PRINT
:
FOR X=1 TO 5
  DEF TAB=X
  PRINT 1,2,3
NEXT X

RUN

1               2               3
1       2       3

1  2  3
1   2   3
1    2    3
1     2     3
1      2      3
```

REMARK Also see TAB, WIDTH, WIDTH LPRINT and PAGE.

DEF USR statement

FORMAT **DEF USR** *digit* = *expression*

DEFINITION The DEF USR statement is used to define the addresses of up to 10 machine language user subroutines; USR0 to USR9.

EXAMPLE *Examples only. Do Not Use!*

```
REM Calls graphic routine at memory address 5000
DEFUSR1=5000
X=USR0(45)
:
DEFUSR2=23445
PRINT USR2(x)
```

REMARK A machine language return is needed at the end of the routine to return program control to ZBasic.

See USR, MACHLG, CALL, LINE, VARPTR, BIN$, HEX$, OCT$, UNS$, PEEK, PEEKWORD, POKE, POKEWORD and the chapter "Machine Language".

Some other default USR functions are included in the appendix for your computer.

WARNING: Use of this command requires a knowledge of machine language and a computer's hardware. Porting of programs with this statement may not be possible without re-writing the routines.

statement **DELAY**

FORMAT **DELAY** *expression*

DEFINITION The DELAY statement will cause a program to pause a specified amount of time.

The *expression* sets the delay in milliseconds; thousandths of a second.

EXAMPLE

```
CLS
FOR I = 1 TO 5
  PRINT "DELAYING  ";I;"SECONDS"
  DELAY  I  *  1000
NEXT  I
END
```

RUN

```
DELAYING 1 SECONDS
DELAYING 2 SECONDS  (after 1 second)
DELAYING 3 SECONDS  (after 2 seconds)
DELAYING 4 SECONDS  (after 3 seconds)
DELAYING 5 SECONDS  (after 4 seconds)
```

```
FOR X=1000 TO 0 STEP -50
  PRINT X
  DELAY X
NEXT
```

(try it)

REMARK The <BREAK> key is not scanned during DELAY. Any negative *expression* will cause delays in excess of 32 seconds (the unsigned value). Note that DELAY -1 will delay over 65 seconds (unsigned -1 = 65,535).

There may be a slight time variation from machine to machine due to processor speed, interupts, hardware differences, etc.

Also see DATE$ and TIME$.

Also see TIMER.

DELETE command

FORMAT

DEL [ETE] *line*
DEL [ETE] *-line*
DEL [ETE] *line - line*
DEL [ETE] *line-*

DEFINITION

This command will remove a line or range of lines from a program in memory.

DELETE is used from the Standard Line Editor.

EXAMPLE

```
10  CLS
20  FOR I  =  1  TO 10
30  PRINT  "NUMBER "; I
40  NEXT  I
50  END

DEL   10-20

LIST

30  PRINT  "NUMBER "; I
40  NEXT  I
50  END
```

```
10  "FRED" PRINT  "NUMBER ";I
20  PRINT "Fred was here"
30  END

DELETE "FRED"

LIST

20  PRINT "Fred was here"
30  END
```

REMARK

Use this command with care as recovery of deleted lines is not possible.

statement **DIM**

FORMAT **DIM** [*len*] *var* [*type*] [(*number* [, *number* ..])] [,...]

DEFINITION The DIM statement is used to allocate memory for variables and array variables and to define common variables for chained programs.

len Defines the length a of a string (how many characters it may hold). This is optional and defines the length of all the following string variables in that DIM statement or until a new length is encountered in that statement. The default is 255 characters unless changed by a previous DEFLEN.

var The name of a variable (any variable type).

type Forces the *variable* to be of that *type* .

%=Integer
&=LongInteger (Macintosh only)
!=Single Precision
#=Double Precision
$=String

Also see "Variables" in the front section of this manual.

number The maximum number of elements that a dimension may contain from 1 to 32,767elements (add one if array BASE option is set to zero. default=0). Only numbers may be used, not variables.

EXAMPLE See the following page for more information and examples.

REMARK Use care when allocating memory with the DIM statement.

See BASE OPTION, DEFLEN, "Array Variables", "String Variables", INDEX$ and RUN+ for more important information about using DIM.

MacIntosh: This version is limited to 2,147,483,648 elements in an array.
MSDOS: In order to optimize performance; integer variables and integer array variables are limited to one 64K segment. String and BCD arrays may cross segment borders to use up to available memory.

continued next page...

DIM statement

DIM continued

DETERMINING THE MEMORY NEEDS OF DIMMED ARRAYS

```
DIM A%(10,10,10), A#(5), A!(9,7), B$(10), 5Cool$(20
DIM Long&(10):  REM  Macintosh Only
```

The following chart shows how to calculate the memory requirements of the arrays dimensioned above with a BASE OPTION of zero.

ARRAY	Type	Bytes per Element	How to Calculate**	Memory Required
A%(10,10,10)	Integer	2	11*11*11*2	2662
A#(5)	Double Precision	8	6*8	48
A!(9,7)	Single Precision	4	10*8*4	320
B$(10)	String	256	11*256	2816
Cool$(20)	String	6	21*6	126
Long&(10)	LongInteger	4	11*4	44

DEFINING STRING LENGTHS WITH DIM

```
DIM X$(10), 20A$, Z$(5), 45TEST$, 10MD$(20,20)
```

In the example above the maximum character capacities are:

X$	255 (default is 255)
A$	20
Z$ (5)	each element of Z$ as 20* (21*5=105 total bytes)
TEST$	45
MD$(20, 20)	each element of MD$(20,20) as 10. (20 * 20 *11=4400 total bytes of memory used)

* If no length is defined, the last given length in that DIM statement is used. In the example each element of Z$(n) gets a length of twenty. If no length is defined in that DIM statement then 255 characters is the default (or the last length used in DEF LEN).

**If you configure BASE OPTION 1 you will not need to add one to the dimension. To calculate the memory required for A%(10,10,10): 10*10*10*2. See "Configure".

Note: Add one to the defined length of each string for the length byte to determine the actual memory requirement of the string. This extra byte is the "Length byte" and it is the first byte in the string. It is what is pointed at by VARPTR(var$).

Important Note: Unpredictable system errors may result if an attempt is made to assign a string variable a string longer then its allocated length. It is also important to define the length of a string at least one greater than the maximum number of characters received in an INPUT or LINEINPUT statement.

command **DIR**

FORMAT **DIR** [*drivespec*]

DEFINITION **DIR** will display the directory of the disk drive specified by *drivespec* .

The *drivespec* will vary from one computer to the next. See your Computer's Disk Operating System reference manual for syntax.

EXAMPLE

```
DIR <ENTER>

LEDGER.COM        MAY.LED       JUN.LED
JUL.LED           AUG.LED

ZBasic  Ready
```

REMARK The appearance of the directory layout will vary by computer. See appendix for further information. This is a command so it does not operate during runtime.

See below, or your appendix, for possible ways of getting directories at runtime.

MacIntosh: Syntax is DIR "rootname or foldername". To get a directory during runtime see FILES$ in the appendix. LDIR will output the directory to a printer.

MSDOS: Use DIR *.BAS to see all the .BAS files or DIR Z*.* to see all the files starting with Z. To get a directory during runtime see FILES.

Apple ProDOS: To get a directory during runtime; OPEN"I" the directory pathname. Example: OPEN"I",1,"ZBASIC". See directory layout in ProDOS reference manual for more information about directory file layout. This version also supports LDIR to list the directory to the printer. CAT may be used as well as DIR.

Apple DOS 3.3: To get a directory during runtime:

```
LONG FN DIR (slot,drive)
  POKE &AA6A,slot
  POKE &AA68, drive
  CALL &A56E
END FN
```

Z-80: See appropriate section in appendix for your computer and DOS. Some Z80 versions do not allow getting a directory at runtime.

DO statement

FORMAT

DO

.

.

UNTIL *expression*

DEFINITION

The DO statement is used to define the beginning of a loop with the UNTIL statement defining the end.

Program functions and statements appearing between the DO and UNTIL will be executed over and over again until the expression defined at the UNTIL statement is TRUE.

EXAMPLE

```
DO
  PRINT"Hi!"
UNTIL LEN(INKEY$)
END

RUN

Hi!
Hi!
Hi!
Hi!                <-----You press a key and it stops
```

```
DO
  X=X+1
UNTIL X=2492
PRINTX
END

RUN

2492
```

REMARK

The statements in a DO loop will be executed at least once. See WHILE-WEND for a loop type that ends immediately if the condition is false.

ZBasic automatically indents text appearing between a DO and UNTIL two spaces. This is helpful in debugging and documenting programs.

See the "Structure" and "Loops" sections of this manual for more information.

Also see FOR-NEXT-STEP and WHILE-WEND.

command **EDIT**

FORMAT

E *line*
EDIT *line*

DEFINITION

EDIT is used from the Standard Line Editor to specify the line you wish to edit.

EDIT may be abbreviated to E. A comma will start editing at the line currently selected by ZBasic's line pointer. List of the EDIT sub-commands:

SUB-COMMAND	DEFINITION
[n]**<SPACE>**	- MOVE CURSOR RIGHT (n characters)
[n]**<BACKSPACE>**	- MOVE CURSOR LEFT (n characters)
I	- Begin INSERT mode at cursor position
X	- Goto the end of the line and EXTEND it
<ESC>	- Exit INSERT mode (you will still be in line edit mode)
[n]**D**	- DELETE characters (if n is used deletes n characters)
[n]**C** key	- CHANGE character to <key> [n] times
H	- HACK to end of line and enter INSERT
[n]**S** key	- SEARCH for [n]th occurrence of <key>
L	- LIST line being edited, home cursor
A	- ABORT changes, restore original line
[n]**K** key	- KILL text to [n]th occurrence of <key>
<ENTER>	- EXIT editing with changes intact
<BREAK>	- ABORT EDIT SESSION (no changes made)

Note: *n* is a number from 1 to 255. If *n* is not used, one is assumed.

EXAMPLE

```
10  FOR I  =  1 TO 20
20    PRINT  I
30  NEXT  I

EDIT  20          <----  or E20 (comma if 20 was the last line used.)

20 _              <----  Press spacebar or backspace to move cursor.
                         Use keys above to edit this line.
```

REMARK

If you want to edit the current line, press the comma key <,> in command mode. It will do the same as E <ENTER>.

Line numbers may be edited in ZBasic. The line being edited will remain unchanged, the edited line with the new line number will be created.

See the "Standard Line Editor" section in the beginning of this manual.

Also see FIND, DELETE, AUTO and LIST.

These versions offer full screen editors as well as the Standard Line Editor. See "Full Screen Editor" in the appropriate appendix for details.

ELSE statement

FORMAT

IF- THEN- **ELSE** *line or label*
IF- THEN- **ELSE** *statement(s)*

DEFINITION

ELSE is used with an IF statement to route control on a false condition.

ELSE may refer to a linenumber or label or it may be followed by one or more statements that will be executed if the condition in the IF statement is FALSE.

EXAMPLE

```
X=99
IF X = 100 THEN STOP ELSE PRINT X
END

RUN

99
```

```
IF X=100 THEN STOP ELSE "End"
END
:
"End"
PRINT"Stopped here."
END

RUN

Stopped here.
```

REMARK

All statements on a line following an ELSE are conditional on that ELSE.

See "Structure", IF-THEN, LONG IF, XELSE and ENDIF.

Also see SELECT CASE.

statement **END**

FORMAT **END**

DEFINITION END is used to stop the execution of a program.

END will return control to the Standard Line Editor if program was executed using RUN, or to the operating system if the program was compiled using RUN* or RUN+.

EXAMPLE

```
PRINT  "HELLO"
END
PRINT  "THERE"

RUN

HELLO
```

REMARK END will close all open files.

Also see STOP and TRONB.

See SHUTDOWN.

END FN statement

FORMAT

LONG FN
.
.
END FN [= *expression*]

DEFINITION

Marks the end of a LONG FN statement.

The optional *expression* MUST be numeric for numeric functions (#,%,&,!) and MUST be a string ($) for string functions.

EXAMPLE

```
REM Removes spaces from the end of a string
LONG FN RemoveSpace$(x$)
  WHILE ASC(RIGHT$(x$,1)=32
    x$= LEFT$(x$, LEN(x$)-1)
  WEND
END FN= x$
Name$="ANDY        "
PRINT "Before:";Name$;"*"
PRINT" After:"; FN RemoveSpace$(Name$);"*"

RUN

ANDY      *         ANDY*
```

```
REM Example of a simple Matrix Multiplication
DIM A%(1000)
:
LONG FN MatrixMult%(number%, last%)
  FOR temp%= 0 TO last%
    A%(temp%)=A%(temp%)*number%
  NEXT
END FN
:
A%(0)=1: A%(1)=2:A%(2)=3
FN MatrixMult%(10,3)
PRINT A%(0), A%(1), A%(2)

RUN

10          20          30
```

REMARK

If an END FN is omitted in a LONG FN construct, a structure error will occur. You must exit a function from an END FN otherwise problems will occur internally.

Also see "Functions and subroutines", "Structure", LONG FN, FN statement, FN function and DEF FN.

Important Note: Loops like FOR-NEXT, DO-UNTIL or WHILE-WEND must be entirely contained within a LONG FN-END FN. Do not exit a function except at the END FN.

statement **END IF**

FORMAT

LONG IF *expression*

.

[XELSE]

.

END IF

DEFINITION

This is an end marker for the LONG IF statement.

Program execution will continue normally at the END IF after completion of a LONG IF or XELSE.

EXAMPLE

```
Love$="Forever"
LONG IF Love$="Forever"
  PRINT "How Romantic!"
XELSE
  PRINT "How heartbreaking!"
END IF
END

RUN

How Romantic!
```

REMARK

If an END IF is omitted in a LONG IF construct, a structure error will occur.

See "Structure", LONG IF, IF-THEN, ELSE and XELSE.

Also see SELECT CASE.

END SELECT statement

FORMAT

```
SELECT [CASE] [expression]
   CASE [IS] relational condition1 [, relational condition] [,...]
      statement(s)
   CASE [IS] condition [, condition] [,...]
      statement(s)
   CASE [IS] boolean expression
      statement(s)
   CASE ELSE
      statement [:statement:...] ]
END  SELECT
```

DEFINITION

END SELECT is the end marker for the SELECT /CASE structure.

When SELECT/CASE is encountered, the program checks the value of the controlling expression or variable, finds the CASE that compares true and executes the statements directly following the CASE statement. After these statements are performed, the program continues at the line after the END SELECT statement:

EXAMPLE

```
A=100
SELECT A
  CASE >100
    PRINT "A>100"
  CASE 100
    PRINT "A=100"
  CASE ELSE
    PRINT"None of the above"
END SELECT
PRINT "Program continues..."
END

RUN

A=100
Program continues...
```

REMARK

Also see SELECT and CASE.

SELECT CASE is not supported with the Z80 versions. See IF and LONG IF for accomplishing the same thing.

SELECT CASE is not supported with this version. See IF and LONG IF for accomplishing the same thing.

function **EOF**

FORMAT **EOF** (*filenumber*)

DEFINITION Returns true if an end-of-file condition exists for *filenumber*, returns zero if the end-of-file has not yet been reached. This function is only available on the Macintosh and MSDOS versions of ZBasic.

EXAMPLE

```
OPEN"I",1,"FILE.TXT"
DO
  LINEINPUT#1, A$
  PRINT A$
UNTIL EOF(1)
CLOSE#1
END
```

What to do if you don't have EOF on your computer:

```
ON ERROR GOSUB 65535 <--- Enable disk error trapping
OPEN"I",1,"FILE.TXT"
IF ERROR GOSUB"Error message"
DO
  LINEINPUT#1, A$
  PRINT A$
UNTIL ERROR <>0
IF ERROR <> 257 THEN GOSUB "Error message"
ERROR=0 <--- Error 257 is an end-of-file error. Reset Error here then continue.
CLOSE#1
END
:
"Error message"
PRINT "A disk error occurred: "; ERRMSG$(ERROR)
INPUT"<C>ontinue or <S>top? ";temp$
IF temp$="C" THEN ERROR=0:RETURN
STOP
```

REMARK Some versions of ZBasic do not support EOF because of system reasons. Also see ERROR function and statement, ON ERROR and ERRMSG$

EOF is not supported on Z80 versions of ZBasic. Use the second example above to accomplish the same thing.

EOF is not supported on the Apple // ProDOS or DOS 3.3 versions of ZBasic. Use the second example above to accomplish the same thing.

ERRMSG$ function

FORMAT **ERRMSG$** (*expression*)

DEFINITION Returns the error message string for the error number specified by expression . In most cases you will use the number returned by the ERROR function when a disk error has occurred.

EXAMPLE

```
OPEN "I",1, "OLDFILE"
ON ERROR GOSUB "Error message"
.
.
.
"Error message"
PRINT "A disk error has ocurred!!"
PRINT "The error was: ";ERRMSG$(ERROR)
ERROR=0:REM ALWAYS SET ERROR TO ZERO AFTER ERROR OCCURS!.
RETURN

RUN

A disk error has ocurred!!
The error was: File Not Found Error in File #1
```

```
FOR X=0 TO 255
  PRINT ERRMSG$(X)
NEXT X

RUN
```

PRINTS ALL THE ERROR MESSAGES FOR THAT COMPUTER.

REMARK ZBasic will display disk errors for you unless you use the ON ERROR disk trapping options.

The ERROR function is commonly used for error trapping and display purposes. The expression is stored as follows:

The low byte is used for the ERROR number	(ERROR AND 255)
The high byte is used for the file number	(ERROR >> 8) or (ERROR/256)

See "Disk Errors", ON ERROR GOSUB and ERROR functions and statements.

ERROR statement

FORMAT **ERROR** [=] *expression*

DEFINITION Allows the programmer to set or reset **ERROR** conditions for the purpose of disk error trapping.

Important Note: If you do the disk error trapping, ERROR must be reset to zero after a disk error occurs or ERROR function will continue to return an error value.

EXAMPLE

```
REM  This routine checks to see if a file exists.  If it
REM  does exist it is opened as random, if it doesn't
REM  exist an error message is returned.
:
LONG FN Openfile%(files$, filenum%, reclen%)
  ON ERROR GOSUB 65535: REM Disk error trapping on
  "Open file"
  OPEN"I",filenum%,file$
  LONG IF ERROR
    LONG IF (ERROR AND 255) <>3
      PRINT@(0,0);"Could not find: ";file$;" Check drive"
      INPUT"and press <ENTER> when ready";temp%
      ERROR=0: GOTO "Open file"
    END IF
  XELSE
    CLOSE# filenum%
  END IF
ON ERROR RETURN: REM Give error checking back to ZBasic
OPEN"R",filenum%, file$, reclen%
END FN
```

REMARK ERROR may also be used as a function. See "Disk Error Trapping", ERROR function, ERRMSG$ and ON ERROR.

Macintosh: Also see SYSERROR in appendix.

MSDOS: See appendix for ways of doing critical error handling.

Apple ProDOS: See appendix for additional ways of trapping ProDOS errors.

function **ERROR**

FORMAT **ERROR**

DEFINITION Returns the number of an ERROR condition, if any.

Zero (0) is returned if no error has occurred.

This function is available to programmers who wish to trap disk errors using the ON ERROR statement.

EXAMPLE

```
ON ERROR GOSUB 65535:REM User disk trapping enabled
OPEN "I",1,"OLDFILE"
IF ERROR=259 GOSUB"NOT FOUND": GOTO 20
ON ERROR RETURN: REM Let ZBasic do the error checking now!
.
.
.
"NOT FOUND"
REM  ERROR 259 is: File Not Found error in Filenumber 1
PRINT" The file is not on that disk!"
PRINT" Please insert the correct disk"
PRINT" and press <ENTER>"
INPUT A$:ERROR=0:RETURN
```

REMARK ERROR may also be used as a statement. See ERROR statement, ERRMSG$ and ON ERROR GOSUB.

Important Note: If you do the disk error trapping, ERROR must be reset to zero after a disk error occurs or ERROR function will continue to return an error value.

Macintosh: Also see SYSERROR in appendix.

MSDOS: See appendix for ways of doing critical error handling.

Apple ProDOS: See appendix for additional ways of trapping ProDOS errors.

function **EXP**

FORMAT **EXP** (*expression*)

DEFINITION Returns e raised to the power of expression. This function is the compliment of LOG. The BCD internal constant of the value of e is:

2.718281828459045235360287471352662497757247093699959 57

The result will be rounded to the digits of precision configured for Double Precision accuracy.

EXAMPLE

```
DEFDBL  A-Z
DO
  INPUT  "ENTER A NUMBER ";X
  PRINT  "e  RAISED TO X =" ; EXP(X)
UNTIL X=0
END

RUN

ENTER A NUMBER  _ 1
e RAISED TO X  =  2.718281828459 <--- 14 digit accuracy
```

REMARK This is a scientific function. See "Configure" for information about configuring scientific accuracy.

For more information about scientific functions see "Math", "Math expressions", "Floating Point Variables", COS, SIN, ATN, TAN, SQR and raise to the power "^".

FILL statement

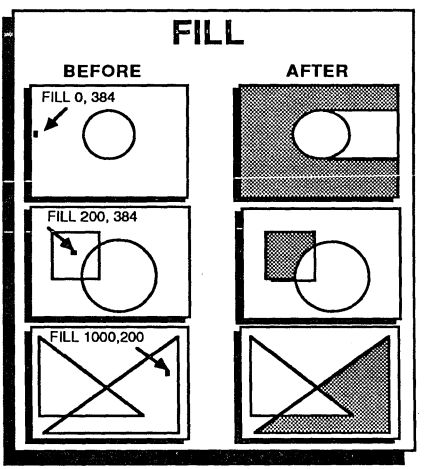
FORMAT **FILL** *expression*$_X$, *expression*$_Y$

DEFINITION The purpose of FILL is to paint an area of the screen in the current COLOR. The point defined by the two expressions are:

expression$_X$ (horizontal position) and *expression*$_Y$ (vertical position).

Fill will search for the uppermost point in the contained area that has the background color, then start filling from left to right and down. For this reason irregular shapes may not fill completely with one fill command. It may be necessary to use a fill statement for each appendage.

EXAMPLE




```
COLOR=1
FILL 0,284
```

RUN

See chart.

REMARK FILL may not be available on machines without the capability of seeing pixels on the screen. See computer appendix. Also see CIRCLE FILL, BOX FILL, MODE, POINT and PLOT.

BOX FILL, CIRCLE FILL and the QuickDraw routines like FILLPOLY, FILLRGN, FILLRECT etc. are much faster ways of filling areas.

command **FIND**

FORMAT

FIND	*commands or keywords*
FIND #	*line*
FIND "	*quoted string text or labels*
FIND REM	*items in REM statements*
FIND DATA	*items in DATA statements*

DEFINITION

FIND is used in the Standard Line Editor to locate text in a program.

To FIND additional occurrences, press semi-colon (;) or FIND <ENTER>.

EXAMPLE

YOU TYPE	ZBASIC FINDS
`FIND "HELLO`	`01010 A=20:PRINT"HELLO THERE"`
`FIND A$`	`01022 Z=1:A$=B$:PRINTA$+B$`
`or...`	`01222 BA$="hello"`
`or...`	`01333 ABA$="goodbye"`
`FIND 99`	`05122 F=2:X=X+2+F/999`
`FIND #12345 (line#)`	`08000 GOTO 12345`
`FIND X(C)`	`03050 A=1:T=ABS(X(C)/9-293+F)`
`or...`	`03044 ZX(C)=4`
`FIND PRINT`	`00230 A=92:PRINTA`
`FIND "SUB5`	`00345 "SUB500": CLS`
`or...`	`03744 GOSUB "SUB500"`
`FIND OPEN`	`03400 OPEN"R",1,"FILE54",23`
`FIND CLOSE`	`09900 CLOSE#2`
`FIND REM This`	`02981 REM This is a remark`
`FIND DATA 123, 232`	`09111 DATA 123, 232`
`FIND DATA "Fred"`	`10233 DATA "Tom", "Dick", "Fred"`

REMARK

When finding a string inside quotes, you must supply all of the characters up to the point that will insure the uniqueness of the string.

See "Standard Line Editor" in the beginning of this manual.

See "Full Screen Editor" in the appropriate appendix for other FIND commands.

FIX function

FORMAT **FIX** (*expression*)

DEFINITION Truncates the digits on the right side of the decimal point.

EXAMPLE

```
PRINT FIX (123.456),
A#=1293.21
PRINT FIX(A#),
PRINT FIX (.12340),
PRINT FIX (999999.455)  + 0.

RUN

123       1293      0         999999
```

REMARK FIX works the same as INT in ZBasic. They are both included to maintain compatibility with other forms of BASIC. FIX will consider an expression floating point.

FRAC is the opposite of FIX. It returns the fraction part of the number.

See FRAC and INT.

function **FN**

FORMAT

FN *name* [(*expression*$_1$ [, *expression*$_2$ [, ...]])]

DEFINITION

FN calls a function by name which was previously defined by DEF FN or LONG FN.

The name of the function must follow the syntax of variable names, that is, a string FN must have a name with a $, an integer FN must have a name with a %, etc.

The *expressions* must match the variable types as defined by the DEF FN or LONG FN. Numeric expressions are not a problem, string expressions allow only simple strings.

FN may not be used before it is defined with DEF FN or LONG FN.

EXAMPLE

```
DEF FN e# = EXP(1.)
DEF FN Pi#= ATN(1) << 2
DEF FN Sec#(x#) = 1.\ COS(x#)
DEF FN ArcSin#(x#) = ATN (x# \ SQR(1-x# * x#))
:
PRINT FN Pi#
```

RUN

```
3.14159... <--- Returned in the current digits of accuracy
```

```
REM Round number to the number of places indicated.
LONG FN ROUND#(number#, places)
  number#=INT(number#*10^places+.5)/10^places
END FN=number#
:
PRINT FN ROUND#(43343.327, 2)
```

RUN

```
43343.33
```

REMARK

This function is useful for saving program space and for making a program easier to read.

Also see "Functions and Subroutines", "Structure", LONG FN, END FN, DEF FN, APPEND and FN statement.

FN statement

FORMAT **FN** *name* [(*expression*$_1$ [, *expression*$_2$ [, ...]])]

DEFINITION FN calls a function by name which has previously been defined by a DEF FN or a LONG FN.

The *expressions* must match the variable types as defined by DEF FN or LONG FN.

EXAMPLE

```
DEF FN LastChr%(x) = PEEK( x + PEEK(x))
LONG FN RemoveSpace$(x$)
   WHILE FN LastChr%(VARPTR(x$)) = ASC("  ")
     x$= LEFT$(x$, LEN(x$)-1)
   WEND
END FN= x$
Name$="ANDY        "
PRINT Name$;"*", FN RemoveSpace$(Name$);"*"


 RUN

 ANDY        *           ANDY*
```

REMARK Also see "Functions and Subroutines", "Structure", LONG FN, END FN, DEF FN, APPEND and FN function.

statement **FOR**

FORMAT

FOR *variable* = *expression*$_1$ **TO** *expression*$_2$ [**STEP** *expression*$_3$]

.
.
.

NEXT [*variable*] [,*variable* ...]

DEFINITION

Permits the repeated execution of commands within the loop.

A FOR/NEXT loop will automatically increment *variable* by the amount set by STEP and compare this to the end value, expression2, exiting the loop when var exceeds this value after adding STEP. Default STEP = 1.

Note the loop will be executed at least once with the value of *expression1* .

EXAMPLE

```
FOR Counter = 0 TO 100 STEP 2
  PRINT Counter;
NEXT

RUN

0 2 4 6 8 10 12 ...  100
```

```
FOR Counter = 100 TO 0 STEP -2
  PRINT Counter;
NEXT Counter

RUN

100 98 96 94 92 90 88 ... 0
```

```
FOR Counter# = 0.0 TO 1.0 STEP .01
  PRINT Counter#;
NEXT Counter#

RUN

0 .01 .02 .03 .04 ... 1
```

REMARK

ZBasic will automatically indent all of its loop structures in listings. This is helpful in debugging and documenting programs.

See chapter called "Loops" and WHILE-WEND and DO-UNTIL.

Note: If STEP is set to zero, the program will enter an endless loop. If the variable is an integer, do not allow the loop to exceed 32,767 or you will enter an endless loop (unsigned integer).

FRAC function

FORMAT **FRAC** (*expression*)

DEFINITION FRAC returns the fractional part of expression. The digits to the left of the decimal point will be truncated.

This function is the compliment of INT and FIX.

EXAMPLE

```
A#=123.456
B#=99343.999
C#=3.5
:
PRINT  A#, FRAC(A#)
PRINT  B#, FRAC(B#)
PRINT  C#, FRAC(C#)
PRINT 2.321, FRAC(2.321)

RUN

123.456        .456
99343.999      .999
3.5            .5
2.321          .321
```

REMARK This function will automatically set floating point calculation.

FIX and INT are the opposite. They return the whole part of the number.

See FIX and INT.

statement **GET**

FORMAT

GET *(x1,y1)-(x2,y2), variable[array(index[,index...,])]*

DEFINITION

Stores a graphic image from the screen into a variable or variable array so that it may be retrieved later and put to the screen with PUT.

GET and PUT are extremely fast and useful for sophisticated graphic animation.

x1, y1 Coordinates of the upper-left-corner of the graphic image on the screen.
x2, y2 Coordinates of the lower-right-corner of the image.

Coordinates are pixel coordinates; use with COORDINATE WINDOW.

The image is normally stored in memory specified by an integer array since it is easier to calculate how much memory is required this way (although other variables may also be used as long as the memory set aside is correct).

To calculate the amount of bytes to DIM for a graphic image, use this equation. Bits-per-pixel (bpp) has to do with colors or grey levels available. See next page for specifics:

6+ ((y2-y1)+1) * ((x2-x1+1) * bpp +7) / 8)

Failure to DIM enough memory for an image will cause unpredictable system errors so be sure to carefully calculate the memory needed.

EXAMPLE

```
DIM A(750)                <--- Bytes above divided by two for integer array
MODE  7                   <--- Not needed on the Macintosh version
COORDINATE WINDOW         <--- Pixel coordinates
:
CIRCLE 100,100,80
GET (0,0)-(100,100), A(1)
:
FOR x= 1 TO 200 STEP 3
  PUT (x, 90), A(1)       <--- Does twice to move the image across
  PUT (x, 90), A(1)            the screen without disturbing the background
NEXT x
:
END
```

This routine moves a section of a circle across the screen. It is PUT to the screen twice so the item doesn't repeat and it will appear to move across the screen without disturbing the background (default PUT *mode* is XOR) .

continued...

GET statement

REMARK

Important Note: Failure to DIM enough memory for the variables storing the graphic images may result in unpredictable system problems.

Also see DIM and PUT.

Macintosh: With this version of ZBasic, PUT has another, optional, parameter: `PUT (x1,y1) [-(x2,y2)], var.` The second parameter allows you to scale the image, making it either larger or smaller by giving the rectangle size in which it is to appear. The x2, y2 parameter is the lower-right corner of the image.

Bits-per-pixel (bpp) will vary by the type of Macintosh you have. The standard black and white Macintoshes have one bit per pixel.

The Macintosh II may have up to 32 bits-per-pixel. Sixteen colors is 4 bpp, 256 colors is 8 bpp. Check addendum or "Inside Macintosh Volume V (Color Quickdraw)" for the specifics of your color board.

MSDOS: Bits per pixel (bpp) will vary by the graphics adaptor board being used:

TYPE	MODE(s)	COLORS	BITS PER PIXEL (bpp)
CGA	5	4	2
CGA	7	2	1
EGA	16-19	3-16	2 (64K or less on EGA card)
EGA	16-19	16	4 (More than 64K on card)
HERCULES	20	1	1

Z80: GET and PUT are not supported with these versions of ZBasic.

Apple // ProDOS and DOS 3.3: GET and PUT are not supported with these versions. See DRAW example on ProDOS disk and the BLOAD and BSAVE functions for possible alternatives.

statement **GOSUB**

FORMAT **GOSUB** *line or label*

DEFINITION GOSUB will call that part of a program starting with line or label and return to the next statement following the GOSUB when RETURN is encountered.

EXAMPLE

```
10 GOSUB 40: PRINT "All Done!"
20 END
30 :
40 PRINT"Hello"
50 RETURN

RUN

HELLO
All Done!
```

```
GOSUB "Hello Routine"
PRINT "All Done!"
END
:
"Hello Routine"
PRINT"Hello"
RETURN

RUN

HELLO
All Done!
```

REMARK On multiple statement lines, a RETURN will return control to the next statement on the line following the originating GOSUB.

To avoid errors, be certain there is a line with the number or label that you GOSUB. All subroutines must be terminated with a RETURN statement.

Note: IF Zbasic encounters a RETURN without a matching GOSUB, it will return to the operating system or the editor. ZBasic does not check for stack overflow which may cause errors if subroutines do not end with a RETURN.

See RETURN LINE, GOTO, ON GOTO and ON GOSUB.

See SEGMENT RETURN in appendix.

GOTO statement

FORMAT **GOTO** *line or label*

DEFINITION **GOTO** will transfer control to a *line or label* in a program.

Note that excessive use of this statement is considered inappropriate for structured code because in complex programs it becomes extremly hard to read.

In most programming situations GOSUB, DO-UNTIL, WHILE-WEND, FOR-NEXT or other programming structures are much easier to follow.

EXAMPLE

```
10 X=X+1
PRINT X,
20 IF X<5 THEN GOTO 10

RUN

1         2         3         4
```

```
"Loop"
X=X+1
PRINT X,
IF X<5 THEN GOTO "Loop"

RUN

1         2         3         4
```

REMARK A line error will occur during compile if the destination line or label cannot be found.

See "Structure", GOSUB, ON GOTO, ON GOSUB, LONG FN, FN statement, WHILE, DO, FOR, LONG IF.

command **HELP**

FORMAT **HELP** [*number*]

DEFINITION HELP without a number prints the HELP menu to the screen. This menu will give you corresponding numbers to the help topics available. This command is used from the Standard Line Editor.

Type HELP and a number to get answers to a specific topic.

Press the SPACE BAR to continue when you see "MORE".

EXAMPLE

```
HELP
```

A menu for your version of ZBasic will be printed to the screen. To get help for an item in the menu, type HELP and the number corresponding to that item.

REMARK HELP will return control to the Standard Line Editor upon completion of the listing.

If the help file has been deleted from the disk a File Not Found Error will occur. Check your computer appendix for the filename of the HELP file.

The HELP window is brought up when you type this command or select "About ZBasic" under the menu. The command does not work exactly as above. Just double click the appropriate item with the mouse.

HEX$ function

FORMAT HEX$ (*expression*)

DEFINITION The HEX$ function converts a numeric expression to a four character HEXadecimal string (BASE 16). The following program will convert a Decimal number to HEX or HEX to Decimal. Some sample HEX numbers:

Decimal	Hexadecimal
0-9	0-9
10	A
11	B
12	C
13	D
14	E
15	F

EXAMPLE

```
DO
  INPUT"Decimal number to convert: ";Decimal%
  PRINT "Decimal";Decimal%;"= HEX ";HEX$(Decimal%)
  PRINT
  :
  INPUT"HEX number to convert: ";Hx$
  Hx$="&H"+Hx$
  PRINT"Decimal value of ";Hx$;"="VAL(Hx$)
  PRINT"The unsigned Decimal value of "Hx$"=" UNS$(VAL(Hx$))
UNTIL (Decimal% =0) OR (LEN(Hx$)=2)

RUN

Decimal number to convert: 255
Decimal 255= HEX FF

HEX number to convert: F9CD
Decimal value of  F9CD = -1587
The unsigned Decimal value of F9CD = 63949
```

REMARK Floating point numbers will be truncated to integers.

See "Numeric Conversions", VAL, OCT$, BIN$ and UNS$.

See DEFSTR LONG in the appendix for doing LongInteger conversions in Hex, Octal, CVI and MKI$. In this case HEX$ would return an eight character string.

statement **IF**

FORMAT

IF *expression* **THEN** *line* [*or label*] [**ELSE** *line* [*or label*]]
IF *expression* **THEN** *statement* [*:statement: ...*] [**ELSE** *statement* [*:statement: ...*]]

DEFINITION

The IF statement allows a program to do a number of things based on the result of *expression*:

1. Branch to a line or label after the THEN if a condition is true; *expression* ≠0
2. Execute statement(s) after the THEN if a condition is true; *expression* ≠0
3. Branch to a line or label after the ELSE if a condition is false; *expression*=0
4. Execute statement(s) after the ELSE if a condition is false; *expression*=0

EXAMPLE

```
X=99
IF X=99 THEN PRINT"X=99":PRINT"HELLO: ELSE STOP
:
IF X=99 THEN "CHECK AGAIN"
END
:
"CHECK AGAIN"
IF X=100 THEN PRINT"YEP" ELSE PRINT"NOT TODAY!";:PRINT X
END

RUN

X=99
HELLO
NOT TODAY! 99
```

REMARK

Complex strings will generate an error if used in an IF statement.

Improper	IF LEFT$(A$,2)="HI" THEN STOP
Proper	B$=LEFT$(A$,2): IF B$="HI" THEN STOP

See LONGIF, ELSE, XELSE , WHILE-WEND and DO-UNTIL for more ways of doing program comparisons.

Note: *In many cases LONG IF is easier to read.*

Also see SELECT CASE.

INDEX$ statement

FORMAT

INDEX$ (*expression*) = *string expression*
INDEX$I (*expression*) = *string expression*
INDEX$D (*expression*)

DEFINITION

INDEX$ is a special array unique to ZBasic. *Expression* indicates an element number.

Statement	Definition
INDEX$(n)=simple string	Assigns a value to INDEX$(n)
INDEX$I(n)=simple string	Move element n (and all consecutiveelements) up and **INSERT** simple string at INDEX$ element n
INDEX$**D**(n)	**DELETE** element n and move all consecutive elements down to fill the space.

EXAMPLE

```
INDEX$(0)="FRED"          <---Normal assignments
INDEX$(1)="TOM"
INDEX$(2)="FRANK"
:
GOSUB"Print INDEX$"
INDEX$I(1)="HARRY"        <----HARRY INSERTED between FRED and TOM
GOSUB"Print INDEX$"
:
INDEX$D(0)                <---FRED is DELETED here
GOSUB"Print INDEX$"
END
:
"Print INDEX$": REM Routine prints contents of INDEX$
FOR X=0 TO 4
  PRINT X; INDEX$(X)
NEXT: PRINT
RETURN

RUN

0 FRED
1 TOM
2 FRANK

0 FRED
1 HARRY
2 TOM           <--- Notice how values move from one element to another
3 FRANK              as items are inserted and deleted with INDEX$I and D.

0 HARRY
1 TOM
2 FRANK
```

REMARK

INDEX$ provides for memory efficient string array manipulation and lends itself very well to list management applications. See "Special INDEX$ Array", INDEX$ function, CLEAR, CLEAR INDEX$ and MEM.

Allows up to ten simultaneous INDEX$ arrays. See INDEX$ in your appendix.

function **INDEXF**

FORMAT **INDEXF** (*string* [,*expression*])

DEFINITION INDEXF is a special INDEX$ array function used to FIND a leading string within an INDEX$ array quickly.

If INDEX$(1000) equaled "Hello", then X=INDEXF("Hel") would return 1000.

If X=INDEXF("llo") X would equal -1 since "llo" would not be found. The leading characters are significant.

EXAMPLE

```
INDEX$(0)="FRED"
INDEX$(1)="MARY"
INDEX$(2)="TOM"
:
X=INDEXF("TOM")          <--- Search for TOM
PRINT X
:
PRINT INDEXF("MARY")     <--- Search for MARY
:
PRINT INDEXF("RED")      <--- Search for RED
:
PRINT INDEXF("FRED",1)   <--- Search for FRED starting at element 1

RUN

 2        <----- TOM found at element two
 1        <----- MARY found at element one
-1        <----- RED not found. The first characters are significant
-1        <----- FRED not found because search started at element 1
```

REMARK INDEX$ provides for memory efficient string array manipulation and lends itself very well to list management and text editing applications.

See "Perpetual Sort" under "Special INDEX$ Array". Also see INDEX$, INDEX$I, INDEX$D , CLEAR, CLEAR INDEX$ and MEM.

Allows up to ten simultaneous INDEX$ arrays. See INDEX$ in your appendix.

INKEY$ function

FORMAT **INKEY$**

DEFINITION INKEY$ returns the character of the last key that was pressed or an empty string if no key was pressed.

EXAMPLE

```
WHILE A$<>"S":  REM Press "S" to Stop
  DO
    A$=INKEY$
  UNTIL LEN(A$)
  A$=UCASE$(A$)
  PRINT A$;
WEND
END
```

RUN

```
GHUIJD,KEUG FAQCCQ OPU... S
```
<---When <S> is pressed program stops

```
REM An easy function you can use to get a key
LONG FN Waitkey$(local$)
  DO
    local$=INKEY$
  UNTIL LEN(local$)
END FN=local$
:
key$=FN Waitkey$(key$)
PRINT key$
END
```

RUN

(user presses "b")

```
b
```

REMARK When using INKEY$ for character entry, avoid having the TRON function active as this may cause pressed keys to be missed.

See INPUT, LINEINPUT, INPUT#, ASC and CHR$. See your computer appendix for variations or enhancements.

Macintosh: See DIALOG (16) for way of doing INKEY$ during event trapping.
MSDOS: INKEY$ returns two characters for function keys. ON INKEY$ does event checking for function keys. See appendix for specifics.

function **INP**

FORMAT **INP** (*expression*)

DEFINITION The INP function is used to read an input port. The function returns the value that is currently at the port specified by *expression.*

EXAMPLE

```
X=INP(1)
PRINT X
PRINT INP(G-1)

RUN

0
255
```

REMARK Note: This function requires a knowledge of your computer hardware and may not be portable to other computers (may not be available on your version of ZBasic or may have an unrelated function).

See your computer appendix for specifics.

Not supported with this version. See INSLOT.

Not supported with this version. See OPEN"C" and "Toolbox" in the appendix for accessing hardware ports.

INPUT statement

FORMAT **INPUT** [(@ or %)(*expr*^{x}, *expr*^{y})] [;] [!] [&*expr*,] [*"string"*;] *var* [,*var* ...]

DEFINITION The INPUT statement is used to input values (string or numeric) from the keyboard into variables.

Multiple variables must be separated by commas (this is bad form since users often forget commas). If no value is INPUT, a zero or null string will be returned.

@(*xpr*^{x},*expr*^{y})	Places cursor at text coordinate horiz,vert.
% (*expr*^{x},*expr*^{y})	Places cursor at graphic coordinate horiz,vert.
;	Suppress carriage return/line feed.
!	Automatic Carriage return after maximum characters entered. User doesn't have to press <ENTER>.
&expr,	Sets the maximum number of characters tobe INPUT. Default is 255. Will not allow more than *expr* characters.
"string";	Optional user prompt will replace the question mark. If a null string is used the question mark will be suppressed.
var	May be any variable type integer, single,double or string.

EXAMPLE *See examples on following pages...*

REMARK Differences in screen width may affect operation.

See LOCATE and PRINT for more information on cursor positioning. Also see INPUT#, LINEINPUT, LINEINPUT# and INKEY$ for others ways of getting input.

See "Keyboard Input" in the technical section.

Important Note: String lengths MUST be one greater than maximum INPUT length since a CHR$(13) is temporarily added. Never define a string used in an INPUT or LINEINPUT as ONE.

In certain cases EDIT FIELD, MENU or BUTTON may be preferable. See appendix.

INPUT continued

EXAMPLES OF REGULAR INPUT

EXAMPLE	RESULT
INPUT A$	Wait for input from the keyboard and store the input in A$. Quotes, commas and control characters cannot be input. <ENTER> to finish. A carriage return is generated when input is finished (cursor moves to beginning of next line).
INPUT"NAME: ";A$	Prints "NAME: " before input. A semi-colon must follow the last quote. A carriage return is generated after input (cursor moves to next line).
INPUT; A$	Same as INPUT A$ above, only the semi-colon directly after INPUT disables the carriage return (cursor stays on the same line).

EXAMPLES OF LIMITING THE NUMBER OF CHARACTERS WITH INPUT

EXAMPLE	RESULT
INPUT &10, A$	Same as INPUT A$ only a maximum of ten characters may be input. (&10) A carriage return is generated after input (cursor moves to the beginning of the next line). The limit of input is set for ALL variables, not each.
INPUT ;&3,I%	Same as INPUT &10, except the SEMI-COLON following INPUT stops the carriage return (cursor stays on line).
INPUT !&10,A$	Same as INPUT & 10 except INPUT is terminated as soon as 10 characters are typed (or <ENTER> is pressed).
INPUT;!&10,"NAME: ";A$	Same as INPUT ;&10,A$ except no carriage return is generated (semi-colon). INPUT is terminated after 10 characters(&10 and Exclamation point). and the message "NAME: " is printed first.
LINEINPUT;!&5,"NAME: ";A$	LINEINPUT A$ until 5 characters or <ENTER> is pressed. (no carriage return after <ENTER> or after the 5 characters are input. Accepts commas and quotes.)

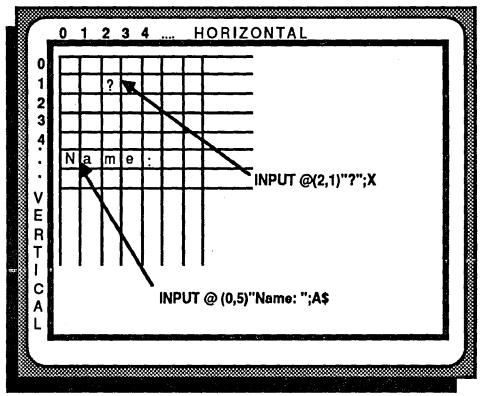
Note 1: Wherever INPUT is used, LINEINPUT may be substituted when commas, quotes or some other control characters need to be input (except with multiple variables).

Note 2: If more than one variable is INPUT, commas must be included from the user to separate input. If all the variables are not input, the value of those variables will be null.

INPUT statement

INPUT continued

INPUTTING FROM A SPECIFIC SCREEN LOCATION



INPUT@(H,V); A$	Wait for input at TEXT screen POSITION defined by Horizontal and Vertical coordinates. No "?" is printed. A carriage return is generated.
INPUT %(gH, gV);A$	Input from a graphic coordinate. Syntax is the same as "@". Very useful for maintaining portability without having to worry about different screen widths or character spacing.
INPUT@(H,V);!10,"AMT: ";D#	Prints "AMT:" at screen position H characters over by V characters down. D# is input until 10 characters, or <ENTER>, are typed in, and the input is terminated without generating a carriage return (the cursor DOES NOT go to the beginning of the next line).
INPUT%(H,V);!10,"AMT: ";D#	Prints "AMT:" at Graphic position H positions over by V positions down. D# is input until 10 characters, or <ENTER>, are typed in, and the input is terminated without generating a carriage return (the cursor DOES NOT go to the beginning of the next line).

Note: Replace INPUT with LINEINPUT whenever there is a need to input quotes, commas and control characters (except with multiple variables).

statement **INPUT#**

FORMAT **INPUT #** *expression* **,** *var* **[** **,***var* **[** **, . . .]]**

DEFINITION This statement will read INPUT from a disk or other device specified by *expression* until a carriage return, <COMMA>, End-Of-File or 255 characters are encountered.

Commas and leading spaces may be read into a string variable if the data on disk was enclosed in quotes, otherwise leading spaces and line feeds will be ignored.

See LINEINPUT# for ways of inputting commas, quotes and some control characters.

EXAMPLE

```
A$="HELLO"
B$="GOODBYE"
C$="WHAT?"
X#=12.345
:
OPEN"O",1,"TEST.TXT":REM OPEN FOR OUTPUT
PRINT#1, A$","B$","C$","X#    <--- Quoted commas important with PRINT#
CLOSE#1
:
OPEN"I",1,"TEST.TXT":REM OPEN FOR INPUT
INPUT#1, X$,Y$,Z$,A#          <--- INPUT# in same order and type as PRINT#
CLOSE#1
:
DEFTAB=10: PRINT X$,Y$,Z$,A#
END

RUN

HELLO    GOODBYE    WHAT?    12.345
```

REMARK See OPEN, CLOSE, PRINT#, and LINEINPUT#.

See your computer appendix for available devices.

Compatibility Note: ZBasic and MSBASIC have almost the same syntax with the following exceptions:

MSBASIC ALLOWS	ZBasic REQUIRES
PRINT#n, A$, B$, X#, C%	PRINT#n, A$","B$","X#","C%
PRINT#n, A$ B$ C$	PRINT#n, A$","B$","C$

If you remember that ZBasic puts the image to the disk just as if it were going to the printer or to the screen you will see why the syntax is important.

INSTR function

FORMAT **INSTR** (*expression* , *string* $_1$, *string* $_2$)

DEFINITION Finds the first occurrence of *string* $_2$ in *string* $_1$, starting the search at the position specified by *expression* .

expression	Starting position of the search.
string1	String to be searched.
string2	String to search for.

EXAMPLE

```
Humble$="I am cool!"
PRINT INSTR(1,Humble$,"cool")
:
B$="am"
PRINT INSTR(1,Humble$, B$)
:
X=INSTR(1, Humble$, "FRED")
PRINT X
END

RUN

6               <--- "Cool" started at the sixth position
3               <--- "am" started at the third position
0               <--- There was no "FRED" in the string.
```

```
Name$="Fred Smith"
Lastname$=RIGHT$(Name$,LEN(Name$)-INSTR(1,Name$, " "))
PRINT "Hello there Mr.";Lastname$
END

RUN

Hello there Mr. Smith
```

REMARK If the string is not found, zero (0) will be returned.

See LEFT$, RIGHT$, MID$ and INDEXF.

function **INT**

FORMAT **INT** (*expression*)

DEFINITION Truncates all digits to the right of the decimal point of *expression* .

EXAMPLE

```
DEFDBL  A-Z
DEFTAB 8
PRINT"       X","ABS(X)","INT(X)","FRAC(X)","SGN(X)"
:
FOR X = -15.0 TO +15.0 STEP 3.75
  PRINT USING"-##.##";X,
  PRINT USING"-##.##";ABS(X),
  PRINT USING"-##.##";INT(X),
  PRINT USING"-##.##";FRAC(X),
  PRINT USING"-##.##";SGN(X)
NEXT X
END
```

RUN

```
   X      ABS(X)     INT(X)     FRAC(X)     SGN(X)
-15.00    15.00     -15.00       .00       -1.00
-11.25    11.25     -11.00      -.25       -1.00
- 7.50     7.50      -7.00      -.50       -1.00
- 3.75     3.75      -3.00      -.75       -1.00
   .00      .00        .00       .00         .00
  3.75     3.75       3.00       .75        1.00
  7.50     7.50       7.00       .50        1.00
 11.25    11.25      11.00       .25        1.00
 15.00    15.00      15.00       .00        1.00
```

REMARK INT works the same as FIX in that expression will be restricted to the integer range of -32,768 to +32,767 only when the expression has not been defined as floating point.

INT is simply as a function that truncates an expression to a whole number.

To get the fractional part of a number use FRAC.

See FIX, SGN, ABS and FRAC.

INT range for the Macintosh is -2,147,483,648 to +2,147,483,647.

KILL statement

FORMAT **KILL** *simplestring*

DEFINITION KILL will erase a disk file specified by simplestring .

KILL functions either as a command or from within a program.

EXAMPLE

```
INPUT"File to erase:";A$
PRINT"Are you sure you want ";A$;" erased?";
INPUT B$
:
LONG IF B$<>"YES"
  PRINT"File not erased": STOP
XELSE
  KILL A$:PRINT A$;" is history."
END IF
:
END

RUN

File to erase: Oldfile
Are you sure you want Oldfile erased?
YES
Oldfile is history!
```

REMARK Use this statement with caution. When a file has been killed it is normally unrecoverable.

See RENAME, ERROR, ON ERROR, ERRMSG$ and the "Files" section of this manual for more information.

This page intentionally left blank.

LEFT$ function

FORMAT **LEFT$** (*string* , *expression*)

DEFINITION LEFT$ returns the left-most characters of *string* defined by *expression*. The string will not be altered.

EXAMPLE

```
Quote$="Early to Bed, Early to rise..."
:
PRINT LEFT$(Quote$, 5)
:
Part$= LEFT$(Quote$, 12)
PRINT Part$
:
PRINT LEFT$(Quote$, 50);
PRINT "Makes men healthy...at least"

RUN

Early
Early to Bed
Early to Bed, Early to rise... Makes men healthy...at least"
```

REMARK Also see RIGHT$, MID$, LEN, VAL, STR$, INSTR, INDEX$, SWAP and the "String Variable" section of this manual for more information about using strings.

function **LEN**

FORMAT **LEN** (*string*)

DEFINITION Returns the number of characters that are stored in a string constant or string variable. If zero is returned it indicates a null (empty) string.

EXAMPLE

```
A$="FRED"
B$="SMITH"
:
PRINT A$;" has";LEN(A$);" characters."
PRINT B$;" has";LEN(B$);" characters."
:
PRINT LEN(A$)+LEN(B$)
:
PRINT LEN("Hello Fred")
```

```
RUN

FRED has 4 characters
SMITH has 5 characters
9
10
```

REMARK The maximum length of a string is 255 characters. You may set the length of strings in ZBasic. See DIM, DEF LEN and the chapter on "String Variables" for more information about defining string length.

Since the first character of a string stored in memory is the length byte, PEEK(VARPTR(var$)) will also return the length of a string.

The memory required for a string variable is the defined length + one for the length byte (256 bytes if not defined).

LET statement

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

DEFINITION LET is an optional statement that may be used to assign an *expression* to a *variable.*

Numbers, strings, numeric *expressions,* or other variables may be used to assign values to a variable if the types are compatible or convertable.

EXAMPLE

```
LET B=100
PRINT B
:
LET B=B+10
PRINT B
:
Z$="HELLO"+" THERE"          <---Notice "LET" is optional
PRINT Z$

RUN

100
110
HELLO THERE
```

REMARK See SWAP, "Optimize expressions for Integer", "Math Expressions" and "Conversions Between Variable Types" for more information about assignments.

function **LINE**

FORMAT LINE *line number or label*

DEFINITION Returns the starting address of a compiled *line* in memory. Normally used with CALL to execute machine language subroutines created with MACHLG.

EXAMPLE

```
10 CALL LINE 30          <--- Example only. DO NOT RUN!
20 END
30 MACHLG 23,323,11,232,A%, 2,1,0,0,1:RETURN
```

```
"START"
PRINT"THIS IS A TEST ",1,2,3
"END"
A = LINE "END"  - LINE "START"
PRINT  "The second line is ";A;" bytes long"

RUN

THIS IS A TEST              1              2              3
The second line is  36 bytes long
```

REMARK This statement is useful for calling machine language subroutines embedded in your program or for calculating the number of bytes used by program lines.

Also see MACHLG and CALL.

MacIntosh: Use LongIntegers for addresses. See CALL in the appendix.

MSDOS: See CALL in appendix.

Apple ProDOS: See MLI in ProDOS appendix.

LINEINPUT statement

FORMAT **LINEINPUT**[(@ or %)(*expr1*,*expr2*)][;][!][&*expr*,][*"string"*;]*var*$

DEFINITION The LINEINPUT statement is used to input characters from the keyboard into a string variable. It is different from INPUT in that quotes, commas and some control characters may also be entered. LINEINPUT is terminated when <ENTER> is pressed.

@(*expr1*,*expr2*) %(*expr1*,*expr2*)	Inputs from horizontal,vertical TEXT coordinate. Inputs from horizontal,vertical GRAPHIC coordinate.
;	Suppresses carriage-return/line-feed after input is complete. (disable inputs that cause scrolling or overwriting.)
!	Automatically executes a carriage return after the maximum number of characters are entered. The user doesn't have to press <ENTER>.
&*expr* ,	Sets the maximum number of characters to be input.
"string";	Optional string prompt will replace the question mark "?" normally shown with LINEINPUT.
var$	Only string variables may be used with LINEINPUT.

EXAMPLE

```
INPUT"Last name <COMMA> First name";A$
PRINT A$
:
LINEINPUT"Last name <comma> First name";B$
PRINT B$

RUN

Smith
Smith, Fred
```

REMARK See the chapter on "Keyboard Input" in the front of this manual for more examples.

The advantage of using LINEINPUT over INPUT is its ability to receive most of the ASCII character set except:

```
<ENTER>          CARRIAGE RETURN
<CTRL C>         CONTROL  "C"
<BACKSPACE>      DELETE or LEFT ARROW
<CANCEL>         DELETE CURRENT LINE
<NULL>           NO CHARACTER
```

Important Note: String lengths MUST be at least one greater than the number of characters being input, otherwise a string overflow condition will destroy subsequent variables. Never use a one character string with LINEINPUT.

statement **LINEINPUT#**

FORMAT **LINEINPUT #** *expression* **,** *variable$*

DEFINITION This statement will input ASCII or TEXT data from a disk file specified by *expression* until <ENTER>, End-Of-File or 255 characters are encountered.

Useful for accepting commas, quotes and other characters that INPUT# will not accept. A good example of using LINEINPUT would be for reading an ASCII or TEXT file a line at a time (as in the example below).

EXAMPLE

```
REM Read a text file and print it to the screen
REM Routine compatible with all versions of ZBasic
:
ON ERROR GOSUB 65535: REM Error trapping on to check for EOF
:
OPEN"I",1,"TEXT.TXT"
:
Counter=0
:
WHILE ERROR=0: REM  Read file until an EOF error
  LINEINPUT#1, A$
  PRINT A$
WEND
IF ERROR <> 257 THEN PRINT ERRMSG$(ERROR): STOP
ERROR=0
:
ON ERROR RETURN: REM Give error trapping back to ZBasic
END
```

REMARK The advantage of using LINEINPUT# over INPUT# is its ability to receive most of the ASCII character set. Leading linefeeds will be ignored on some systems.

If a CHR$(0) or CHR$(26) is encountered as a leading character it may assume EOF and set ERROR = End Of File (varies by computer).

Also see INPUT#, LINEINPUT and "Keyboard Input" in the front section of the manual.

These versions support an EOF function that would simplify the error trapping techniques used above. See the appropriate appendix for details about EOF:

```
OPEN"I",1,"TEXT.TXT"
Counter=0
:
WHILE EOF=0: REM  Read until EOF
  LINEINPUT#1, A$
  PRINT A$
WEND:CLOSE#1
```

LIST command

FORMATS

[L]L [IST] [+][*]
[L]L [IST] [+][*] *line or label*
[L]L [IST] [+][*] *- line or label*
[L]L [IST] [+][*] *line or label - line or label*

DEFINITION

LIST (or L) is used from the Standard Line Editor to list the current program to the screen. LLIST will list the current program to a printer.

+ Suppress line numbers
* Highlight keywords on the screen (some versions)

EXAMPLE

YOU TYPE	ZBASIC RESPONDS
LIST or L	Lists complete program to the screen
LLIST	Lists complete program to the printer
LIST 100-200	Lists lines from 100-200
LLIST-100	Lists lines up to 100 to printer
LIST "SUBROUTINE"	Lists the line with that label
LIST 100- or L100-	Lists lines from 100 on
<period>	Lists the last line listed or edited
<UP ARROW>	Lists previous line (or plus <+> key)*
<DOWN ARROW>	Lists next line (or minus <-> key)*
L+	Lists program without line numbers
LLIST+	Lists to printer without line numbers
L+-100	Lists up to line 100 without line numbers
<SPACE>	PAUSE. <ENTER> continues
</> (slash key)	PAGE AT A TIME: Lists 10 lines to the screen*

*See computer appendix for keyboard variations.

REMARK

LIST automatically indents program lines two spaces between FOR-NEXT, DO-UNTIL, WHILE-WEND, LONG IF-XELSE-END IF and LONG FN-END FN structures.

See PAGE, WIDTH, WIDTH LPRINT and the chapter; "Formatting listings".

Note: Labels may be used in place of line numbers.

LLIST+* will format listings to an Imagewriter or Laserwriter with no line numbers and with keywords in bold. While the output in of this format is extremely attractive and easy to read, it should be noted that listings will take about twice as long to print.

command **LOAD**

FORMATS

LOAD ["] *filespec* ["]
LOAD* ["] *filespec* ["]

DEFINITION

LOAD is used from the Standard Line Editor to load a ZBasic tokenized or a regular ASCII text file into memory.

ZBasic does not load tokenized files from other languages; the file must first be saved in TEXT or ASCII format.

If the program does not have line numbers they are added in increments of one .

LOAD* will strip away remarks and unnecessary spaces from an ASCII file releasing more room for the source and object code in systems with limited memory.

EXAMPLE

```
LOAD    PROGRAM      <--- Loads a regular tokenized or text file
LOAD    "SOURCE"     <---Double Quotes optional
LOAD*   THISONE      <---Strips spaces and REM's while loading
```

REMARK

Each operating system may require specific syntax for a drivespec.

Line numbers are optional in ASCII files.

If a program was created using another form of BASIC it must be in ASCII format before the ZBasic editor can load it.

These versions of ZBasic support a Full Screen Editor that may support other forms of LOAD. See appropriate appendix for information about full Screen Editors.

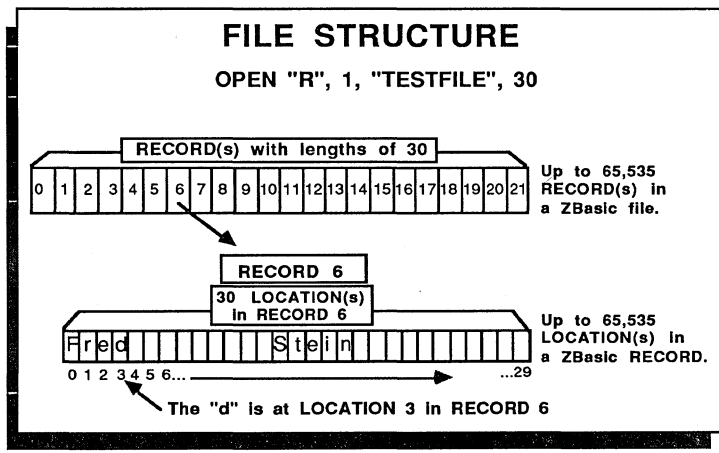
LOC function

FORMAT **LOC** (*expression*)

DEFINITION Returns the byte pointer position within the current RECORD of the filenumber specified by *expression* .

EXAMPLE

```
OPEN"R",1,"TESTFILE",30
RECORD#1, 6, 3                <--See illustration
PRINT LOC(1)
:
READ#1, Char$;1
PRINT LOC(1)
:
PRINT Char$
CLOSE#1

RUN

3
4
d
```



REMARK The LOC position is incremented to the next file position automatically when READ#, WRITE#, INPUT#, LINEINPUT# or PRINT# are used. REC(filenumber) returns the current RECORD. LOF returns the last record in the file. Also see "Files" section for more information.

The record length limits are different for these versions. See appendix.

statement **LOCATE**

FORMAT **LOCATE** $expr_x$, $expr_y$, [$expr_{cursor}$]

DEFINITION Positions the cursor to the coordinates given by *exprx*, *expry* and optionally turns on or off the cursor character (zero=off, not zero=on).

$expr_x$	The horizontal coordinate (characters across)
$expr_y$	The vertical coordinate (lines down)
$expr_{cursor}$	Zero= cursor OFF. Non-zero = cursor ON

EXAMPLE

```
LOCATE 0,0          <---- sets cursor in upper left corner
LOCATE 10,0         <---- sets Cursor 10 char to right at top
LOCATE 0,10,0       <---- sets Cursor 10th line down. Cursor OFF
LOCATE 0,12,1       <---- sets Cursor 12th line down. Cursor ON
```

REMARK This function is also useful with CLS LINE and CLS PAGE for clearing the screen to the end of line and end of page .

See "Screen and Printer Control", PRINT@, PRINT%, INPUT@, LINEINPUT@, LINEINPUT% and INPUT% for other ways of controlling the cursor positioning.

The ability to turn the cursor on or off may be limited by the hardware or software of some computers.

These versions of ZBasic allow you swap the horizontal and vertical coordinates under "Configure". This is handy for converting other BASIC programs that use the vertical coordinate first (not Apple DOS 3.3).

LOF function

FORMAT **LOF** (*expression*)

DEFINITION Returns the last valid RECORD number for the file specified by *expression.* LOF stands for Last-Of-File.

Important Note: This function may not return the last record correctly on some systems, especially if the record length of the file is different from the operating system's internal record length or if a file is opened with a different record length then that which it was opened originally. This is often remedied by simply setting the record length to the system default record length or the record length of which it was opened originally.

EXAMPLE See "Opening files for Append" in the "Files" section in the front of this manual for methods of getting a pointer to the last position in a file.

REMARK LOF returns the last record in the file. The default record length is 256 and may need to be changed to make LOF function properly.

See LOC and REC for getting file pointer information. See "Files" and "Disk Errors" for more information. Some systems return one for both record zero and record one.

Note to better usage: If you need to keep track of the last byte position of a sequential file or the last record of a random file , you might consider storing the last REC and LOC of a file in record zero before it is closed. Examples:

```
OPEN"O",1,"Textfile.txt"
RECORD#1,1 <---Set file pointer to record one (zero will store last REC and LOC)
PRINT#1,A$","B$","X","Z# <--- Save data
RECORD#1,0<---- Position pointer to RECORD 0 to save last REC and LOC
R=REC(1):L=LOC(1)
WRITE#1, R,L <--- Save pointers for future use
CLOSE#1
```

To add data to the end of the file later:

```
OPEN"R",1,"Textfile.txt"
RECORD#1,0
READ#1, R, L     <--- Get last positions of file
RECORD#1, R,L    <----- Position pointer to append data to the end of the file.
PRINT#1, A$      <--- Now you can append new data to the file
```

Don't forget to store the LOC and REC before closing! You could do the same thing with random files by saving the last record.

Also supports: LOF(*filenumber*, [*recordlength*]). LOF(1,1) would return the length of filenumber one in bytes.

function **LOG**

FORMAT **LOG** (*expression*)

DEFINITION Returns the natural logarithm of expression (LN). LOG is the compliment of EXP.

Common LOG10= LOG(n) \LOG(10)

EXAMPLE

```
PRINT LOG(2)
X#=LOG(3)
PRINT X#

RUN

.69314718056
1.09861228857
```

REMARK LOG is a scientific function. Scientific precision may be configured by the user differently from both single and double precision.

See "Configure" and "Math" in the beginning of this manual.

Also see COS, SIN, EXP, "^", ATN and TAN.

LONG FN statement

FORMAT

LONG FN *name* [(*var* [, *var* [, ...]])]
.
.
END FN [= *expression*]

DEFINITION

LONG FN is similar to DEF FN but allows the function to span over several lines. This is usful for your own functions that you can use with ZBasic.

A re-usable, non-line-numbered function may be saved to the disk with SAVE+ and retrieved later for use in other programs with APPEND.

The variables being passed to the function must not be arrays. The *expression* must be numeric for numeric functions and string for string functions.

EXAMPLE

```
LONG FN RemoveSpace$(x$)
  WHILE ASC(RIGHT$(x$),1)=32
    x$=LEFT$(x$,LEN(x$)-1)
  WEND
END FN= x$
:
Name$="ANDY          "
:
PRINT Name$;"*"
:
Name$=FN RemoveSpace$(Name$)
PRINT Name$;"*"

RUN

ANDY          *
ANDY*
```

```
REM Wait until key press. Return key in key$
LONG FN WaitKey$(key$)
  DO
    key$=INKEY$
  UNTIL LEN(key$)
END FN=key$
:
Z$=FN WaitKey$(Z$)
PRINT Z$

RUN
```

(returns key that was pressed)

REMARK

Also see APPEND, SAVE+, DEF FN, FN statement, FN function and "Structure".

statement **LONG IF**

FORMAT

LONG IF *expression*

.

[XELSE]

.

ENDIF

DEFINITION

LONG IF allows multiple line IF-THEN-ELSE structures. Very useful for breaking down complicated IF statements into more readable, logical structures. Two things happen based on the result of expression:

- If expression is TRUE: Executes all the statements up to the XELSE (if used) and then exits at the END IF.
- If expression is FALSE: Executes all the statements between the XELSE and END IF and then exits at the END IF. If XELSE is not used it will simply exit at the END IF.

EXAMPLE

```
INPUT"How old are you: ";Age%
LONG IF Age% >=30
  PRINT "You are Old aren't you!?"
XELSE
  PRINT "You're just a baby!"
END IF

RUN

How old are you: 30
You are Old aren't you!?
```

```
LONG IF Name$="Fred"
  PRINT"Hello Fred...Long time no-see!"
  PRINT"The balance you owe is";USING"$####.##";Due#
  PRINT"Thanks for asking."
XELSE
  PRINT "I don't know you! Go away!"
END IF

RUN

Hello Fred...Long Time no-see!"
The balance you owe is $1234.56
Thanks for asking.
```

REMARK

No loop may be executed within a LONG IF construct unless it is completely contained between a LONGIF and XELSE or between the XELSE and ENDIF. The entire LONG IF construct must be completely contained within loops or nested loops in order to compile properly.

ZBasic will automatically indent program lines between LONG IF, XELSE and END IF two spaces. See the chapter about "Structure" for more information.

LPRINT statement

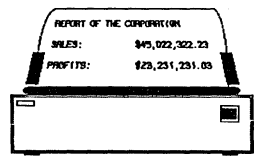
FORMAT **LPRINT** *[variables, constants,...]*

DEFINITION The LPRINT statement sends output to a printer.

To use LPRINT from the Standard Line Editor use a colon first (:LPRINT).

EXAMPLE

```
LPRINT "REPORT OF THE CORPORATION"
LPRINT
LPRINT
LPRINT "SALES:";TAB(50);USING"$##,###,###.##";Sales#(1)
LPRINT
LPRINT "PROFITS:";TAB(50);USING"$##,###,###.##";Profits#(1)

RUN
```



REMARK Some systems may lock up if a printer is not connected. See your hardware manual for required action.

See ROUTE 128, PRINT, LLIST, TAB, DEFTAB, PAGE, USING, WIDTH LPRINT and POS(1).

MacIntosh: See DEF LPRINT, PRCANCEL, DEF PAGE, PRHANDLE, TEXT and ROUTE 128 in the appendix for more information about printing to the Imagewriter and Laserwriter printers. See appendix for specifics.

MSDOS: To use more than one printer you may also use OPEN"I",1,"LPT2:" and use PRINT#1, [variables, constants...]. Be sure to close the printer device when finished. See MSDOS reference manual for more information about LPT2:, LPT1: and any other devices you may have available for your hardware.

Apple ProDOS and **DOS 3.3:** See DEF LPRINT for setting the printer slot.

statement **MACHLG**

FORMAT **MACHLG** { [*bytes* ,...]} -or- { [*words* ,...] } -or-{ [*variables*] [, ...] }

DEFINITION The MACHLG statement is used to insert bytes directly into a compiled program. These bytes may be machine language programs, variables or other items.

It may be used to insert machine language into memory without using POKE.

bytes	Numbers from 0 to 255
words	Numbers from 0 to 65535. They are stored in standard format
variables	Will create the address where the variable is located. See appendix for specifics.
	Note: ZBasic uses registers when calculating elements of an array variable. Contents of these registers may be destroyed.

EXAMPLE

```
X = LINE "Machine Language Routine"
FOR I = 0 TO 10
  PRINT PEEK(X+I);
NEXT I
END
:
"Machine Language Routine"
MACHLG 0,1,2,3,4,5,6,7,8,9,10

RUN

0 1 2 3 4 5 6 7 8 9 10
```

REMARK

See LINE, CALL, USR, DEFUSR, PEEK, POKE and the chapter about "Machine Language" in the technical section of this manual.

Important Note: Use of this statement requires knowledge of the machine language of the computer you are using. Machine language may not be portable to other computers.

Macintosh: Since the Macintosh is a 32 bit machine, MACHLG puts the code into word, not byte, positions.

MSDOS: See DEF SEG in appendix.

Apple ProDOS: See section entitled Machine Language Interface in appendix.

MAYBE function

FORMAT **MAYBE**

DEFINITION MAYBE is a random function that returns either a TRUE (-1) or FALSE(0) with equal probability.

MAYBE is faster than RND, convenient, and requires little program space.

EXAMPLE

```
DEFTAB = 8: DIM Coin$(1)
Coin$(0)="HEADS":Coin$(1)="TAILS"
:
"Flip a Coin"
DO
  X=X+1
  PRINT Coin$(MAYBE+1),
UNTIL X=25
END
```

RUN

```
HEADS     HEADS     TAILS     HEADS     TAILS
TAILS     TAILS     TAILS     HEADS     HEADS
TAILS     TAILS     HEADS     TAILS     TAILS
HEADS     HEADS     HEADS     HEADS     TAILS
HEADS     TAILS     TAILS     TAILS     HEADS
```

REMARK This function is useful anytime a 50% random factor is needed.

MAYBE with logical operators:

MAYBE	50% TRUE	50% FALSE
MAYBE **AND** MAYBE	25% TRUE	75% FALSE
MAYBE **OR** MAYBE	75% TRUE	25% FALSE

command **MEM**

FORMAT **MEM**[ORY]

DEFINITION Typing either MEM or MEMORY in command mode will return information about system memory use.

TEXT	The number of bytes being used by the source code. The source code is that part of the program that you type in.
MEMORY	The number of bytes remaining for program use (varies; see your computer appendix for details).
OBJECT	The size of the object code after compiling. *Valid only immediately after RUN.*
VARIABLES	The number of bytes required for variables, INDEX$ array, and disk I/O buffers. This varies dramatically by version. See computer appendix. *Valid only immediately after RUN.*

EXAMPLE

```
MEM

00046  Text
41244  Memory
00039  Object
00388  Variable
(some versions may display more information)
```

REMARK These numbers are relative to that version of ZBasic being used. Varies significantly by computer.

See your computer appendix for more information.

Also see MEM function, CLEAR, CLEAR INDEX$, CLEAR END, LOAD* and the chapter about "Converting Old Programs".

MEM function

FORMAT **MEM**

DEFINITION Returns the number of bytes available in the INDEX$ array.

EXAMPLE

```
CLEAR  1000
PRINT MEM
A= MEM
INDEX$(0)   = STRING$(49,"*")
PRINT MEM

RUN

1000
950
```

REMARK See also INDEX$, MEM command, and CLEAR INDEX$. This function varies by version. See appendix for specifics.

MEM(*index number*) returns the memory available to that INDEX$ (there are ten available on the Macintosh).

MEM (-1): Returns the maximum amount of memory available for variables. Also forces unloading of all unlocked memory segments. Returns a LongInteger.

INDEX$ has many enhancements with this version. See appendix.

See appendix for various additions to the MEM function that return memory pointers to arrays, strings, BCD variables and more.

command **MERGE**

FORMATS

MERGE ["] *filespec* ["]
MERGE * ["] *filespec* ["]

DEFINITION

MERGE is used to overlay a line numbered TEXT/ASCII program from disk onto the current program text in memory. Program being merged must be in ASCII (saved with SAVE*).

Incoming text with the same line number(s) as resident text will replace resident text.

The asterisk is used to strip spaces and REM's from the incoming program.

EXAMPLE

```
010 REM Program one
120 DO
130   I$=INKEY$
140 UNTIL LEN(I$)
SAVE*  "PROG1"
NEW

10 REM Program two
20 PRINT "MAIN MENU"
30 PRINT
40 PRINT "1. Do Inventory"
50 PRINT "2. Print Inventory"
60 PRINT "3. Delete Inventory"
MERGE  "PROG1"
LIST

00010 REM Program one    <---- Line from first program overwrote this line
00020 PRINT "MAIN MENU"
00030 PRINT
00040 PRINT "1. Do Inventory"
00050 PRINT "2. Print Inventory"
00060 PRINT "3. Delete Inventory"
00120 DO                 <--- First program merged here
00130   I$=INKEY$
00140 UNTIL LEN(I$)
```

REMARK

MERGE has the same affect as manually typing in text.

Programs that were written in another BASIC must be in ASCII format before being MERGED into ZBasic.

Also see LOAD , SAVE*, RENUM, APPEND and DELETE

MID$ function

FORMAT MID$ (*string* , *expr1* [, *expr2*])

DEFINITION Returns the contents of *string* starting at position *expr1*, and *expr2* characters long.

string	The string from which the copy will occur.
expr1	The distance from the left that the copy will begin.
expr2	Optional parameter that determines how many characters will be copied. If omitted, all characters from expr1 to the end of the string will be copied.

EXAMPLE

```
A$="The Sun Shines Bright"
:
PRINT MID$(A$,5,3)
:
Z$=MID$(A$,15)
PRINT Z$
:
FOR Pointer = 1 TO LEN(A$)
  PRINT MID$(A$,Pointer,1)
NEXT

RUN

Sun
Bright
T
h
e

S
u
n
.
.
.
```

```
INPUT"First and Last name please:";Name$
PRINT "Thank you Mr. ";MID$(Name$,INSTR(1,Name$," ")+1)

RUN

First and Last name please: Fred Smith
Thank you Mr. Smith
```

REMARK See LEFT$, RIGHT$, INSTR, LEN, STR$ and the MID$ statement.

statement **MID$**

FORMAT MID$ (*string1* , *expr1* [, *expr2*]) = *string*$_2$

DEFINITION Replace a portion of *string1* starting at *expr1*, with *expr2* characters of *string2.*

string1	Target string. String2 will be inserted or layed over this string.
string2	String to be inserted or layed over string1.
expr1	Distance from the left of *string1* where overlay is to begin
expr2	How many characters of *string2* to insert into *string1*. Using 255 will assure that all characters are used.

EXAMPLE

```
A$  =  "SILLY  BOY"
B$  =  "SMART"
:
PRINT A$
:
MID$(A$,1,5) = B$
PRINT A$

RUN

SILLY BOY
SMART BOY
```

REMARK This function is very useful for altering selected portions of strings.

Also see RIGHT$, LEFT$, MID$ function, STR$, INSTR, VAL, LEN, SPACE$, STRING$.

MKB$ function

FORMAT **MKB$** (*expression*)

DEFINITION Returns a string which contains the compressed floating point value of a ZBasic BCD *expression* .

This function works with either single or double precision. The amount of string space used will vary depending on the digits of precision configured. See "Configure".

To return the floating point values stored in strings use the CVB function.

EXAMPLE

```
A$=MKB$(991721.645643)
PRINT "The length of A$=";LEN(A$)
X!=CVB(A$)
PRINT X!
:
PRINT
:
B$=MKB$(991721.645643)
PRINT "The length of B$=";LEN(B)
X#=CVB(B$)
PRINT X#

RUN

The length of A$=4      <--- Value returned depends on configured precision
991722

The length of B$=8      <--- Value returned depends on configured precision
991721.645643
```

REMARK Since ZBasic automatically compresses and decompresses BCD variables when using READ# and WRITE#, this function is of primary interest to those people that need to conserve memory for other reasons.

See also CVB, CVI, READ#, WRITE# and MKI$.

See your appendix for default accuracy and variations.

function **MKI$**

FORMAT MKI$ (*expression*)

DEFINITION Returns a two character string which contains a two byte integer specified by *expression* .

To extract the integer stored in a string wth MKI$, use the CVI function.

EXAMPLE

```
A$=MKI$(12345)
PRINT"Length of A$=";LEN(A$)
B%=CVI(A$)
PRINT B%
PRINT
:
A$=STR$(12345)
PRINT "Length of A$=";LEN(A$)
PRINT VAL(A$)

RUN

Length of A$=2
 12345                    <--- MKI$ saves space...(4 bytes compared to below)

Length of A$=6
 12345                    <--- Leading blank reserved for the "SIGN"
```

REMARK Used in older versions of BASIC to convert integers to strings for FIELD statements. ZBasic does this automatically when using READ# and WRITE#. Nevertheless, MKI$ and CVI are still useful for packing strings to save memory-- especially on systems with limited memory.

See also CVI, CVB, READ#, WRITE# and MKB$.

Use DEFSTR LONG to allow MKI$, CVI, HEX$, OCT$ and BIN$ to work with LongIntegers. Use DEFSTR WORD to set back to regualr integer. Note that MKI$ returns a four byte string with LongIntegers.

MOD operator

FORMAT *expression*$_1$ **MOD** *expression*$_2$

DEFINITION MOD returns the remainder of an integer division with the sign of *expression*$_1$.

EXAMPLE

```
PRINT "9 DIVIDED BY 2=";INT(9/2);"REMAINDER =";9 MOD 2
RUN
9 DIVIDED BY 2= 4 REMAINDER= 1
```

```
PRINT "-4 DIVIDED BY 2=";INT(-4/2);"REMAINDER=";-4 MOD 2
RUN
-4 DIVIDED BY 2= -2 REMAINDER= 0
```

REMARK MOD replaces the old BASIC routines for finding the remainder of a division and is also much faster:

OLD BASIC:	X = (X - INT(X/N) * N)
ZBasic:	X = X MOD N

statement **MODE**

FORMAT **MODE** *expression*

DEFINITION MODE is used to set the screen graphics or text format.

Most computers offer a number of different character and/or graphic modes. Use MODE to choose the mode most applicable to the program.

For most systems EVEN modes are character graphics and ODD modes are regular graphics. Not all machines have graphic capability. MODE for some popular microcomputers:

Mode number	MSDOS type		APPLE //e, //c		TRS-80 I, III	
	Text	Graphic	Text	Graphic	Text	Graphic
0	40x25	character	40 x 24	character	32x16	character
1	40x25	40x40	none	40x48	64x16	128x48
2	80x25	character	80x24	character	32x16	character
3	80x25	80x25	none	80x48	64x16	128x48
4	80x25	character	40x24	character	32x16	character
5	40x25	320x200	40x24	280x192	64x16	128x48
6	80x25	character	80x24	character	32x16	character
7	80x25	640x200	80x24	560x192	64x16	128x48
8	40x25	character	40x24	character	32x16	640x240?
9	40x25	40x40	Bottom	40x48	64x16	128x48
10	80x25	character	80x24	character	32x16	character
11	80x25	80x25	Bottom	80x48	64x16	128x48
12	80x25	character	80x24	character	32x16	character
13	40x25	320x200	Bottom	280x165	64x16	128x48
14	80x25	character	80x24	character	32x16	character
15	80x25	640x200	Bottom	560x165	64x16	128x48

MACINTOSH	
Text	Graphic
Many Font styles and sizes here!	SEE Macintosh APPENDIX

CP/M-80	
Text	Graphic
Normally 80x24	SEE Z80 APPENDIX

Be sure to read the appropriate appendix for exact mode designations.

REMARK MODE will reset COLOR to the default, usually the darkest background and lightest foreground, and may clear the screen with some systems.

Macintosh: MODE is ignored with the Macintosh. See the TEXT statement for setting character styles and sizes. To emulate other computers you will probably want to use Monaco or Courier mono-spaced fonts. TEXT *font, size, face, mode.*

MSDOS: Modes 16-19 support EGA modes. Mode 20 supports Hercules graphics. See appendix for details.

MOUSE function

FORMAT **MOUSE** (*expression*)

DEFINITION Returns information concerning the position and status of a MOUSE or JOYSTICK if one is connected to the system. The following values are returned.

MOUSE (0)	Initializes the MOUSE on some systems (itialization is required on the Apple // ProDOS and DOS 3.3 versions).
MOUSE (1)	Returns the horizontal coordinate of the mouse.
MOUSE (2)	Returns the vertical coordinate of the mouse.
MOUSE (3)	Returns 0 if button not pressed. Non-zero if button pressed.

EXAMPLE

```
MODE 5 :REM GRAPHIC MODE
CLS
X= MOUSE  (0)                      <---Initialize mouse
:
WHILE LEN(INKEY$)=0                <--- Press any key to stop
  LONG IF MOUSE (3)                <--- If button down then ok to draw
    PLOT MOUSE (1), MOUSE (2)      <--- Plot where mouse (or joystick) is
  END IF
WEND
```

REMARK The above example uses a mouse to draw on the screen. A joystick may also be used (depending on the system). See your computer appendix for hardware device specifics that may apply to these functions.

Also see DEF MOUSE.

Macintosh Note: You may use the mouse functions above or configure ZBasic for MSBASIC Mouse compatibility using DEF MOUSE=1. See Mac Appendix.

MSDOS: Compatible with Microsoft Mouse. ZBasic has to be configured to support a mouse. See "Configure" in MSDOS appendix. If MOUSE(0) <> 0 then a mouse is installed. MOUSE(3) returns 0-3; Zero if both buttons up, three if both buttons down, one or two if one button pressed. MOUSE(4) and MOUSE(5) hide and show the mouse cursor. DEF MOUSE=0 for Mouse, 1 or 2 for joysticks, 3 for lightpens.

Apple ProDOS and DOS 3.3: Compatible with AppleMouse or joysticks. Use DEF MOUSE=0 for AppleMouse or DEF MOUSE=1 for Joysticks. If using a joystick MOUSE(3) returns 0-3. Zero if both buttons up, three if both buttons down, one or two if one button pressed. See appendix for specifics.

Z80: MOUSE IS NOT SUPPORTED with Z80 versions of ZBasic.

statement **NAME**

FORMAT **NAME** *string1* **AS** *string2*

DEFINITION Renames a file with a filename of *string1* to *string2*. Same as the RENAME statement except for syntax. This statement is provided to make ZBasic compatible with other BASIC languages.

EXAMPLE

```
DIR

FRED.BAS                TOM.BAS
DICK.BAS                HARRY.BAS

NAME FRED.BAS AS GEORGE.BAS

DIR

GEORGE.BAS              TOM.BAS
DICK.BAS                HARRY.BAS
```

REMARK See RENAME for more information.

Not available on Apple // or Z80 versions of ZBasic. See RENAME.

NEW command

FORMAT **NEW**

DEFINITION NEW is used to clear the text buffer of the current program.

Since programs that have been erased in this manner are impossible to recover, *SAVE your program first!*

EXAMPLE

```
LIST+

CLS
PRINT"THIS IS A PROGRAM  ';
PRINT"WHICH IS ABOUT TO BE LOST FOREVER AND EVER..."
END

NEW
LIST
```

(Nothing listed...)

REMARK Use this command with care. See LOAD.

statement **NEXT**

FORMAT

FOR var = *expression1* TO *expression2* [STEP *expression3*]

.

.

NEXT [*variable* ,[*variable* . . .]]

DEFINITION

The NEXT statement is used as the end marker of a FOR loop. There must be a matching NEXT for every FOR, otherwise a Structure Error will occur at compile time.

EXAMPLE

```
FOR Count1= 1 TO 2
  FOR Count2 = 2 TO 4 STEP 2
    PRINT Count1, Count2
NEXT Count2, Count1

RUN

1          2
1          4
2          2
2          4
```

```
FOR X= 1 TO 2
  FOR Y= 1 TO 2
    PRINT X,Y
  NEXT
NEXT

RUN

1          1
1          2
2          1
2          2
```

REMARK

The *variable(s)* following the NEXT statement are optional; however, if used they must match the corresponding FOR *variable(s).*

A FOR-NEXT loop will execute AT LEAST ONCE!

A Structure Error will specify the line number if there is an extra NEXT; or will specify line 65535 if a NEXT is missing. ZBasic automatically indents all loop structures when you LIST your program. This may be used to find where the missing NEXT is located by simply following the program listing back to the point where the extra indent ends.

See "Loops" in the front of this manual and; WHILE-WEND, DO-UNTIL, LONGIF-XELSE-ENDIF for other loop and structure types.

NOT operator

FORMAT **NOT** *expression*

DEFINITION NOT returns the opposite of *expression*. True is False, False if True. This is equivalent to changing a logical true (-1) to a logical false(0) and vice versa.

With Boolean (binary) operations, the NOT function will toggle all bits in *expression*. That is, all bits that are one will be changed to zero, and all bits that are zero will be changed to one.

EXAMPLE

```
A$="Hello"
IF NOT A$="Bye" THEN PRINT"True, it is False"
END

RUN

True, it is False
```

REMARK A logical true is -1 and logical false is 0. Also see XOR, OR, AND.

NOT condition TRUE(-1) if condition FALSE, else FALSE (Ø) if TRUE

NOT	
NOT	1 = Ø
NOT	Ø = 1

BOOLEAN "16 BIT" LOGIC			
NOT	11ØØ11ØØ	NOT	Ø1111Ø11
=	ØØ11ØØ11	=	1ØØØØ1ØØ

Will also function with 32 bit LongIntegers.

function OCT$

FORMAT OCT$ (*expression*)

DEFINITION OCT$ returns a 6 character string which represents the Octal value (base 8) of the result of *expression* truncated to an integer. Octal digits are from 0-7.

OCTAL	DECIMAL equivalent
0-7	0-7
10	8
11	9
12	10
13	11
14	12
15	13
16	14
17	15
20	16

EXAMPLE The following program will convert a decimal number to Octal or an Octal number to decimal:

```
CLS
DO
  INPUT"Decimal number: ";Decimal%
  PRINT "Octal Equivalent: ";OCT$(Decimal%)
  :
  INPUT"Octal number: ";Octal$
  Octal$="&O"+Octal$
  PRINT"Decimal Equivalent: ";VAL(Octal$)
UNTIL (DECIMAL%=0) OR (LEN(Octal$)=2)

RUN

Decimal number: 8
Octal Equivalent: 000010

Octal number: 100
Decimal Equivalent: 80
```

REMARK Conversions are possible from any base to any other base that ZBasic supports.

See the Chapter "Numeric Conversions" in the front of this manual. See also BIN$, HEX$ and UNS$.

Use DEFSTR LONG if you want to use OCT$, HEX$, BIN$, UNS$, MKI$ or CVI with LongIntegers. Use DEFSTR WORD to set back to regular integer.

ON ERROR statement

FORMAT

ON ERROR GOSUB *Line or label*
ON ERROR *RETURN*
ON ERROR GOSUB *65535*

DEFINITION

The ON ERROR allows the user to enable and disable disk error trapping. If ON ERROR is not used ZBasic will display disk errors as they occur and give the user the option of continuing or stopping. Options offered with ON ERROR:

ON ERROR GOSUB *65535*	Enable user disk error trapping. Errors are returned using the ERROR function. You must check for errors---ZBasic will not when this parameter is set.
ON ERROR GOSUB *line*	If a disk error occurs the program does a GOSUB to the line or label specified.
ON ERROR RETURN	Disable user disk error trapping. ZBasic will trap the disk errors and give error messages at runtime.

EXAMPLE

```
ON ERROR GOSUB 65535: REM Enable disk error trapping
"Start"
OPEN "I" ,1, "TEST"
IF ERROR GOSUB"Disk error"
GOTO "Start"
program continues...
:
:
"Disk error"
LONG IF (ERROR AND 255)=3: REM Check for File not found error
  PRINT"Check that correct diskette is in drive: <ENTER>";
  DO
  UNTIL LEN(INKEY$)
  ERROR=0:RETURN
XELSE
  PRINT"A Disk Error has occurred:";ERRMSG$(ERROR)
  PRINT"<C>ontinue or <S>top?";
  DO
    temp$=UCASE$(INKEY$)
  UNTIL (temp$="C") OR (temp$="S")
  IF temp$="C" THEN ERROR=0: RETURN
END IF
PRINT"Program aborted!"
ERROR=0
STOP
```

REMARK

Also see ERROR and ERRMSG$ and the chapter about "Disk Error Trapping" in the "Files" section of the manual.

See RETURN *line* for another way of returning from ON ERROR GOSUB *line*.

Important Note: Always remember to set ERROR=0 after a disk error occurs when you are doing the disk error trapping. Failure to do this will cause ZBasic to continue to return a disk error condition.

statement **ON GOSUB**

FORMAT

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

DEFINITION

The ON GOSUB statement is used to call one of several subroutines depending on the value of *expression.*

The ON statement will call the first subroutine if the expression evaluates to one, to the third subroutine if the expression evaluates to three and so on.

The RETURN statement at the end of a subroutine will return the program to the statement immediately following the ON GOSUB.

EXAMPLE

```
"Inventory Menu"
CLS
PRINT "1. Inventory"
PRINT "2. Print Listing"
PRINT "3. Month End"
PRINT "4. EXIT
PRINT
PRINT "Enter item wanted: ";
:
DO
  Item%=VAL(INKEY$)
UNTIL (Item% >0) AND (Item% <5)
:
ON Item%  GOSUB  "Inventory","Print","EOM","Exit"
GOTO "Inventory Menu"
END
:
"Inventory"
RETURN
:
"Print"
RETURN
:
"EOM"
RETURN
:
"Exit"
END
```

REMARK

ZBasic will truncate *expression* to an integer. For example, if *expression* equalled 1.9, the ON statement would go to the first line (INT(1.9)=1).

If *expression* <=0 or > (number of line numbers listed), the program will continue on to the next statement in the program.

ON GOTO statement

FORMAT **ON** *expression* **GOTO** *line* [, *line* [, *line* . . .]]

DEFINITION The ON GOTO statement is used to branch , or jump, to one of several portions of a program depending on the value of *expression* .

The ON statement will jump to the first subroutine if the expression evaluates to one, to the third subroutine if the expression evaluates to three, and so on.

EXAMPLE

```
A=RND(4)
ON A GOTO "ONE", "TWO", "THREE", "Last"
END
:
"ONE"
PRINT 1
END
:
"TWO"
PRINT 2
END
:
"THREE"
PRINT 3
END
:
"Last"
PRINT 4
END

RUN

4
```

REMARK ZBasic will truncate *expression* to an integer. For example, if expression equalled 1.9, the ON statement would go to the first routine (INT(1.9)=1).

If *expression* <=0 or > (number of line numbers listed), the program will continue on to the next statement in the program.

See "Structure".

statement **OPEN**

FORMAT

OPEN "I", [#] *filenumber*, *filename* [, *record length*]
OPEN "O", [#] *filenumber*, *filename* [, *record length*]
OPEN "R", [#] *filenumber*, *filename* [, *record length*]

DEFINITION

The OPEN statement is used to access a data file. Once a file is opened, information may be read from or written to the file depending on the way the file was opened. The first argument determines access:

"R"	Read/write file: Open file if it exists, create the file if it doesn't.
"I"	Read only file: Open file for input. If file doesn't exist, a disk error occurs (file not found error).
"O"	Write only file: Open file for output. Overwrites the old file.
filenumber	The number you assign to a file which is subsequently used with file commands like READ#, WRITE#, INPUT#, LINEINPUT#, PRINT#, REC, LOC and LOF.
filename	The filename as it appears in a directory. See your DOS manual and the appendix in this manual for information about drive specifiers, pathnames, sub-directories or whatever syntax is used for that computer.
record length	Optional record length to be used with that file (default is 256).

EXAMPLE

```
REM Open a file for READ and WRITE
OPEN  "R",1,"INVEN", 180
:
REM Open a file for Input only
OPEN  "I", File%, D$+"INVEN", 180
:
REM Open a file for Output only
OPEN  "O",2, Filename$
```

REMARK

To configure ZBasic to have more than two files open at a time; see "Configure". Each file buffer will require between 160 and 1024 bytes of memory depending on the Disk Operating System and your version of ZBasic. No more than 99 files may be open at one time.

See your computer appendix for more information about file types, changing directories and more. Also see INPUT#, PRINT#, READ#, WRITE#, LOC and REC.

TO INSURE DATA INTEGRITY, ALWAYS CLOSE OPEN FILES BEFORE EXITING YOUR PROGRAM.

continued...

OPEN statement

OPEN continued

 **Macintosh**: Extra parameters included:

volume%	The number you get from FILES$ that sets the folder or root location of the file. Much easier than pathname specifiers. See appendix for details. Also see FILE$, EJECT, EOF, LOF, "File size", APPEND and pathnames. Example of volumn number: `OPEN"type", fnum, "filename", 200, volume%`
Additional types	"R[R]", "O[R]", "I[R]", "A[R]" and "R[D]", "O[D]", "I[D]", "A[D]" The optional "R" or "D" after the file type specifies opening the resource fork (R) or data fork (D). The data fork is the default. See appendix for specifics. The "A" type opens a file for append. Also see APPEND for positioning the file pointer to the end.
Pathnames	Pathnames are supported like: `Root:Folder:Fred`

 **MSDOS**: There are many ways to specifiy, create or remove directories and sub-directories. See PATH$, CHDIR, MKDIR and RMDIR in the appendix .

 **Apple ProDOS**: See PATH. Filenames may contain pathname information like: `PROFILE/ZBASIC/SOURCE`. See appendix for details.

Apple DOS 3.3 uses CP/M type drivespecs like: A: instead of D1, B: instead of D2, etc. Filetype is specified by a leading exclamation mark and a number:

OPEN"-", filenumber, "[[***!type***] [***drivespec***] filename", record length

!type=		
	1= Text file	5= S type file
	2= Integer BASIC	6= Relocatable file type
	3= Applesoft BASIC	7= A type file
	4= Binary file	8= B type file

Example: `OPEN"-", fnum, "!4 A:Fred", 200`

 **CP/M-80**: You may use a drive specifier in the filename:
`OPEN"-",n,"A:Fred.DAT", 200`

TRS-80: You may use a drive specifier in the filename :
`OPEN"-",n,"Fred/DAT.password:1",200`

statement **OPEN "C"**

FORMAT **OPEN "C"**,-1 or -2 [,[*baud rate*][,[*parity*][,[*stopbit*][,*word length*]]]]

DEFINITION This statement is used to set serial communication port parameters. If any of the parameters are omitted the default will be used.

-1	Serial port one
-2	Serial port two
baud rate	110, 150, 300(default), 600, 1200, 2400, 4800, 9600
parity	0 = none <-- default 1 = odd 2 = even
stopbit	0 = one <-- default 1 = two
word length	0 = 7 bits 1 = 8 bits <-- default

EXAMPLE

```
REM A Very Cheap Terminal Program
OPEN"C",-1, 300                  <---Change parameters as needed
DO
  READ#-1, A$;Ø                  <--- (;Ø) Won't "Hang" if nothing at port
  IF LEN(A$) THEN PRINT A$;
  :
  A$=INKEY$
  IF LEN(A$) THEN PRINT#-1,A$;
UNTIL A$="]"                     <--- Set a key to stop
```

REMARK Serial ports may be accessed using the same statements used in disk I/O: PRINT#, INPUT#, LINE INPUT#, READ#, and WRITE#. In all of these statements, the port is not read or written to until the status indicates that the port is ready.

The one exception to the paragraph above is when READ# is used to read a string of zero length . In this case, the character will be returned if ready, otherwise a null string will be returned (similar to the INKEY$ function) (*Not supported with CP/M*).

A port does not have to be opened in order to be accessed. The OPEN "C" statement is used only to set the current port parameter values. Without this statement, the port will simply use the parameters to which it was last set.

All versions have a number of machine specific parameters. See appendix for important details.

continued...

OPEN "C" statement

OPEN "C" continued

The following are examples of sending or receiving files over a modem or serial line. Check appendix and hardware manuals for specifications.

Add your own line numbers, and modify programs as needed. Save with SAVE+ to use later.

SEND FILES TO ANOTHER COMPUTER

```
"SEND FILES"
LINEINPUT"File to send: ";File$
IF LEN(File$)=Ø THEN STOP: REM  No file? STOP
:
OPEN"I",1,File$
ON ERROR GOSUB 65535: REM Catch errors
:
OPEN"C",-1,300: REM Change parameters as needed
:
DO
  LINEINPUT#1, Line$
  IF LEN(Line$) THEN PRINT#-1, Line$
  DO                <---- This DO loop is an example of "Handshaking" remove
    READ#-1,A$;0          this loop , and the PRINT# below, if not needed.
  UNTIL ASC(A$)=1
UNTIL ERROR
:
IF ERROR=0
CLOSE#1
PRINT#-1,"*END*": REM Tell receiver "All Done!"
RETURN
```

RECEIVE FILES FROM ANOTHER COMPUTER

```
"RECEIVE FILES"
LINEINPUT"Filename to Receive: ";File$
IF LEN(File$)=Ø THEN STOP: REM No File? STOP
:
OPEN"O",1,File$
:
OPEN"C",-1,300: REM Change parameters as needed
:
DO
  LINEINPUT#-1, Line$
  IF Line$<>"*END*" THEN PRINT #1, Line$
  PRINT#-1, CHR$(1);   <--- Goes with "Handshaking" Do Loop above.
UNTIL (Line$="*END*")
:
CLOSE#1
RETURN
```

operator **OR**

FORMAT *expression* **OR** *expression*

DEFINITION Performs a logical OR on the two *expressions* for IF THEN testing and BINARY operations. If either or both conditions are true the statement is true. See truth table below.

In binary/boolean operations if either bit is one than a one is returned.

EXAMPLE

```
A$="HELLO"
IF A$="GOODBYE" OR A$="HELLO" THEN PRINT"YES"

RUN

YES
```

REMARK Truth table for the OR function.

condition **OR** condition TRUE(-1) if either or both is TRUE, else FALSE(Ø)

OR		BOOLEAN "16 BIT" LOGIC		
1 OR 1 = 1		ØØØØØØØ1		1ØØØØ1Ø1
Ø OR 1 = 1	OR	ØØØØ1111	OR	1ØØØØ111
1 OR Ø = 1	=	ØØØØ1111	=	1ØØØØ111
Ø OR Ø = Ø				

Also see AND, XOR and NOT.

Functions with 32 bit LongInteger as well.

OUT statement

FORMAT **OUT** *port, data*

DEFINITION The OUT statement sends data to the specified port number.

EXAMPLE

```
OUT  1,12
:
A=6:B=9
OUT A,B
:
OUT  A/2,B/3
END
```

REMARK This statement is microprocessor dependent and works only with Z80 and 8086 type processors.

Also see INP for a way of reading data in from the port.

Not supported with these versions.

statement **PAGE**

FORMAT **PAGE**

DEFINITION Returns the current line position of the printer. The first line is line zero.

EXAMPLE

```
PAGE                    <---Also see PAGE statement
PRINT PAGE
LPRINT
LPRINT
LPRINT
PRINT PAGE

RUN

0
3
```

REMARK This function is similar to POS except the line position is returned instead of the character position.

Important Note: If your operating system uses forms control and checks lines per page, you must disable the operating systems forms control or ZBasic's PAGE.

See CSRLN in the MSDOS appendix for getting the line position of the screen cursor.

PAGE function

FORMATS **PAGE** [[*expression1*] [, [*expression2*] [, [*expression3*]]]]

DEFINITION **PAGE** is used to format output to the printer and to control the number of actual lines per page, printed lines per page and top margin. Following is a description of the parameters:

PAGE	Without parameters will send a page feed to the printer. this forces the print head to move to the defined position of the top of the next page.
expression1	The number of printed lines per PAGE.
expression2	The number of *actual* lines per PAGE. Also resets line count to zero (normally 66 lines per page).
expression3	Lines for the top margin. This number is a subset of expression1. If the line count is zero, this many linefeeds will be output immediately.

EXAMPLE

```
PAGE 60,66,3        <---  Sets Listings to 60 lines per page
                          with 3 lines as top margin. Skips perforations nicely.
```

REMARK WIDTH LPRINT should be set to your printer's character width for proper PAGE operation when doing LLIST.

See PAGE function.

To disable PAGE use PAGE Ø.

Important Note: If your operating systems uses forms control and checks lines per page, you must disable the operating systems forms control or ZBasic's PAGE.

PATH

FORMATS **PATH**

DEFINITION PATH or PATH type commands are available on many versions of ZBasic that support multi-level directories. Rather than give the exact syntax for each machine this page gives a general overview. See your computer appendix for specifics.

MSDOS See PATH$ function in the appendix. This allows you to get the current path name so that you can return to that sub-directory. Syntax is PATH$(drive number). Note: Drive A=1, B=2, ...

Pathname syntax example: C:\ZBasic\TEMP

Apple ProDOS See PATH command in the appendix. Also see the example function on the master disk called : PREFIX.SAMPLE for ways of getting ProDOS pathnames during runtime.

Pathname syntax example: /PROFILE/ZBASIC/OBJECT

Pathnames not supported with DOS 3.3 version.

MacIntosh The most appropriate way of specifying where a file is located is using the volumn number. This is recommended in "Inside Macintosh". Volume numbers are obtained easily using the FILES$ function. See Macintosh appendix.

Nevertheless, pathnames are supported and may be used.

Pathname syntax example: Fred:Tom:Harry

Z80 Pathnames are not supported since the operating systems for this CPU do not currently implement sub directories.

EXAMPLE See your appendix for examples.

REMARK This command varies significantly by computer type.

See DIR, OPEN and also be sure to see your appendix for specifics.

Pathnames are not supported with Apple DOS 3.3 or Z80 versions of ZBasic.

PEEK function

FORMAT

PEEK [WORD] (*expression*)
PEEK LONG (*expression*)*

DEFINITION

Returns the contents of the memory location(s) specified by *expression* :

PEEK	Returns a one byte number (0-255)
PEEK WORD	Returns a two byte number (-32768 to 32767)
PEEK LONG*	Returns a four byte number (*32 bit versions)

EXAMPLE

```
X=VARPTR(A$)              <---Get a safe place in memory to play with
:
POKE X, 10
POKE WORD X+1, 12000
:
PRINT PEEK(X)
PRINT PEEK WORD(X+1)

RUN

10
12000
```

REMARK

See POKE, POKE WORD and POKE LONG, USR, MACHLG, CALL, LINE, HEX$, OCT$, UNS$ and the section in the front of this manual; "Machine Language".

Important Note: This function is for people experienced with machine language and the hardware of their computer.

***Macintosh**: Always use LongIntegers for *expressions* to pass an address or to retrieve a four byte LongInteger. See appendix.

MSDOS: An extra parameter is available to determine the segment of the variable: PEEK[WORD] (*address, segment*). Also see MEM and DEF SEG in the appendix.

statement **PLOT**

FORMAT

PLOT *expr1 , expr2* [**TO** *expr3, expr4 ...*]
PLOT [TO] *expr1 , expr2* [**TO** *expr3, expr4 ...*]

DEFINITION

The PLOT statement is used to draw either one graphic point, or a line between two or more points, in the current COLOR. Examples:

```
PLOT 10,12                    <-- PLOT one point at position 10,12
PLOT 10,12 TO 100,100         <-- PLOT a line from 10,12 to 100,100
PLOT 10,12 TO 10,90 TO 1,1    <-- PLOT two lines: 10,12 to 10,90, to 1,1
PLOT TO 10,12                 <-- PLOT a line from last position to 10,12
```

EXAMPLE

```
CLS
MODE 5              <---Set graphics mode
PLOT  209, 304      <--- Plots one pixel
:
COLOR -1            <--- Sets COLOR to foreground
REM PLOT an angle
PLOT 209,304 TO 987, 643 TO 322,742
END
```

RUN

See illustrations on the following page.

REMARK

As with all other ZBasic graphic commands, Device Independent Graphic coordinates of 1024 by 768 are the default. Expressions are truncated to an integer. Character type graphics will be substituted on computers, or modes, without graphic capabilities.

Also see CIRCLE, BOX, FILL, POINT, COLOR.

Macintosh: Use COORDINATE WINDOW to set to pixel graphics. Use COORDINATE to set yur own relative coordinates or to set back to 1024x768. The upper left-hand corner of a WINDOW is coordinate 0,0.

MSDOS: Use COORDINATE WINDOW to set pixel coordinates. See COORDINATE to set relative coordinates or to set back to ZBasic coordinates.

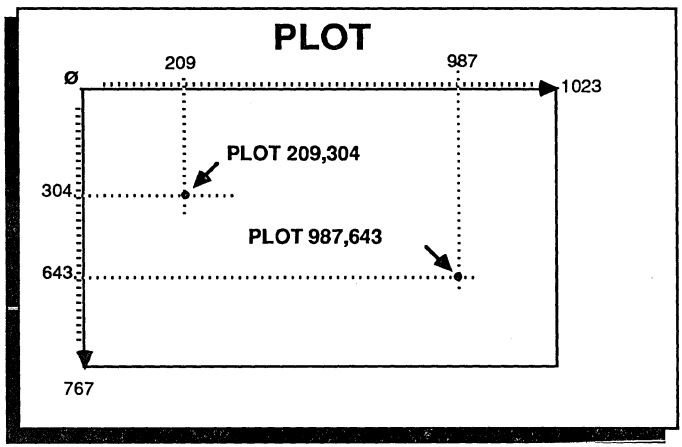
Z80: POKE &xx3F, &C9 for pixel coordinates. POKE &523F, &C3 to set back to ZBasic coordinates. xx= CP/M=01, TRS-80 model 1,3=52. TRS-80 model 4=30.

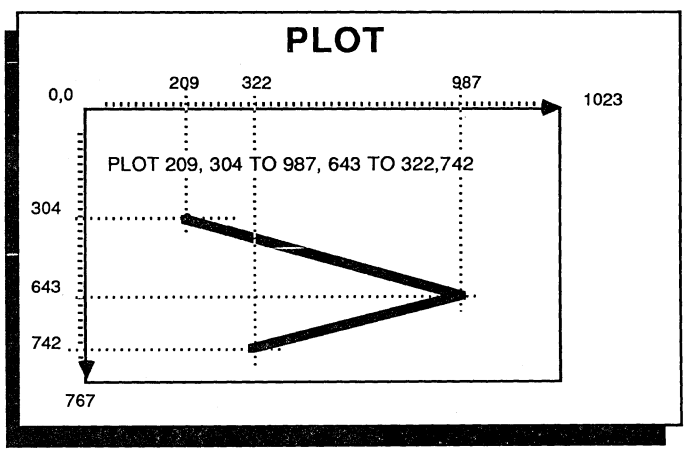
Apple // ProDOS: POKEWORD &85, 0 for pixel coordinates. Use MODE to set back to ZBasic coordinates.

Apple // DOS 3.3: POKE &F388,&60 for pixel coordinates. POKE &F388, &A9 to set back to ZBasic coordinates.

PLOT statement

PLOT continued



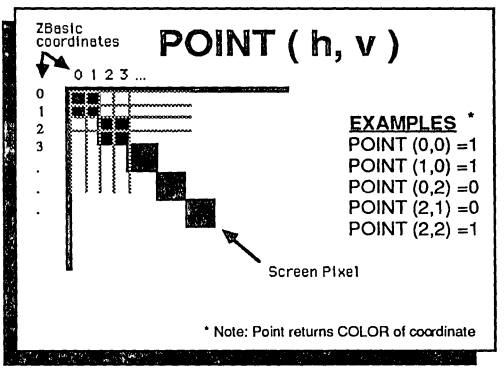


function POINT

FORMAT **POINT** (*expression*$_1$, *expression*$_2$)

DEFINITION Point is available on many computers to inquire about the COLOR of a specific screen graphic position. As with other commands, ZBasic *Device Independent Graphic* coordinates may overlap pixels.

In the example: Ø=Background (white here), 1 =Foreground (black here)



As with all other ZBasic graphic commands, the device independent coordinate system of 1024 x 768 is the default.

EXAMPLE

```
COLOR 1
PLOT 0,0 to 900,767
PRINT POINT(0,0)

RUN

1
```

REMARK If the coordinate is outside screen coordinates, a -1 will be returned.

See COLOR, BOX, CIRCLE and the section; "Graphics".

See COORDINATE or PLOT for ways of converting some versions of ZBasic to pixel coordinates that can used with POINT.

POINT is not available for CP/M versions (including the Kaypro graphic versions).

POKE statement

FORMAT

POKE [WORD] *expression%, expression2*
POKE LONG *expression&, expression2&**

DEFINITION

POKE writes the value of *expression2* into a memory location. The first *expression* is the address to POKE. The *expression2* is the data to POKE.

TYPE	*expression2*
POKE	One byte
POKE WORD	Two bytes
POKE LONG*	Four bytes (*32 bit machines only)

EXAMPLE

```
X = 12345: XA = VARPTR(X)
PRINT"Byte at ";UNS$(XA);" =";PEEK(XA)
:
POKE XA,99
PRINT"Byte at ";UNS$(XA);" =";PEEK(XA)
:
POKE WORD XA,44444
PRINT"WORD at ";UNS$(XA);" =";UNS$(PEEK WORD(XA))
END

RUN

Byte at 59009  =  57
Byte at 59009  =  99
Word at 59009  =  44444
```

REMARK

Also see PEEK, PEEK WORD, PEEK LONG, MACHLG, CALL, LINE and the chapter "Machine Language" at the beginning of this manual.

Important Note: Indiscriminate use of this command may cause unpredictable computer operation and loss of data or program. This statement is for experienced machine language programmers only. Porting of programs with POKE is not recommended.

***Macintosh**: Always use LongIntegers for addresses and when using POKE LONG or PEEK LONG.

MSDOS: There is an optional parameter for segment:
POKE [WORD] *address, data,* ***segment***. See MEM and DEF SEG in the appendix.

function **POS**

FORMAT **POS** (*byte expression*)

DEFINITION Returns the current horizontal cursor position, from zero to 255, for a screen, printer or disk file.

The *expression* specifies a device as follows:

POS(0)	Default device (normally the video monitor)
POS(1)	Printer
POS(2)	Disk file (limited to one file using carriage returns)

EXAMPLE

```
CLS
PRINT  "READ  and  DISPLAY SCREEN POS"
FOR  I  =  0  TO  30 STEP  10
  PRINT TAB(I);  POS(0)
NEXT
PRINT
:
PRINT  "READ  and  DISPLAY PRINTER POS"
DEFTAB  5
FOR  I  =  0  TO  6
  LPRINT,
  PRINT POS(1),
NEXT
END
```

```
RUN

READ  and  DISPLAY SCREEN POS
0          10          20              30

READ  and  DISPLAY  PRINTER  POS
6      12      18      24      30      36
```

REMARK A carriage return will set the POS value to zero. PAGE will return the current line position for the printer.

Also see WIDTH, PAGE and WIDTH LPRINT.

While this command will work the same on all systems, it is dependent on screen and printer widths.

PRINT# statement

FORMAT **PRINT #** *expression* **,** *list of things to print.....*

DEFINITION Used to PRINT information to a disk file or other device in text format. Numbers or strings will appear in the file or device similar to how they would look on the screen or printer.

The *expression* is the file number assigned to a disk file or other device in an OPEN statement.

INPUT# or LINEINPUT# are normally used to read back data created with PRINT# (although READ# may also be used).

EXAMPLE

```
A$="TEST":B$="TEST2":C=900
:
OPEN  "0" ,1, "TEST.DAT"
PRINT#1,"HELLO"","A$","B$","C <--- Quoted comma delimeters for INPUT#
CLOSE#1
:
OPEN"I",1,"TEST.DAT"
INPUT#1, X$, Y$, Z$, A%        <--- INPUT in same order and same type
:
PRINT X$, Y$, Z$, A%
:
CLOSE  #1
END

RUN

HELLO      TEST       TEST2      900
```

REMARK While this command will work the same on all systems, it is dependent on disk input/output capabilities. Use INPUT# or LINEINPUT# to read back data written with PRINT#.

Be sure to see the entry on INPUT# in this reference section for more information about using PRINT# and INPUT# together and also information about MSBASIC syntax differences.

See ROUTE, OPEN, OPEN"C", INPUT#, LINEINPUT#, READ#, WRITE#, LPRINT and the section in the front of this manual called "Files" for more information.

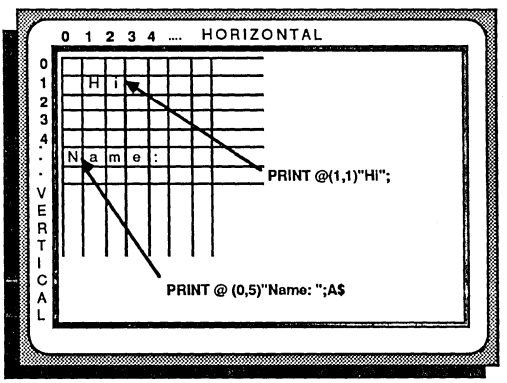
statement **PRINT**

FORMAT **PRINT** [{@ | %} (*expr1* , *expr2*)] [*list of things to print....*]

DEFINITION The PRINT statement is used to output information to the current device, normally the video.

@ (expr1,expr2)	Specifies text coordinates.
% (expr1,expr2)	Specifies graphic coordinates.
	Note: Expr1=Horizontal. Expr2=Vertical.

EXAMPLE




```
PRINT@(1,1)"Hi";
PRINT@(0,5)"Name:";A$
END
```

REMARK PRINT followed with a semi-colon will disable the carriage return.

A PRINT item followed by a comma will cause the next element to be printed at the next tab stop defined by DEF TAB.

While this command will work the same on all systems, it is dependent on hardware.

See ROUTE for ways of sending PRINT data to another device like a printer, disk file or serial port.

See "Screen and Printer Text Control" in the front section of this manual for other ways of formatting text.

As with all other ZBasic graphics commands, PRINT %(x,y) defaults to printing at the position specified by the Device Independent Graphic coordinates of 1024 x 767. See PLOT or COORDINATE for ways of changing some versions of ZBasic to using other coordinates.

PRINT USING function

FORMAT **PRINT[#** *filenumber* **,] USING** *formatstring* **;** *numeric expression* **; [USING...]**

DEFINITION This function permits formatting numeric data in PRINT or PRINT# satements.

The last numeric digit displayed will be rounded up by adding 5 to the first digit on the right that is not displayed.

The *formatstring* may be a quoted or string variable using the following symbols:

Symbol	Definition
#	Holds place for a digit. More than one may be used. An example of using this symbol to hold dollars and cents: `PRINT USING "$###.##";A#` `$123.45`
,	Insert a comma in that place. An example of using it to format numbers with dollars and cents would be: `PRINT USING"$##,###.##";A#` `$12,345.67`
.	Determines placement of decimal point within the format field: `PRINT USING"$##,###,###.##";A#` `$12,345,678.90`
$	Prints a dollar sign on the left of the format. See examples above.

+	Prints a floating plus or minus sign on the side of the number where the plus sign holds the place.
	`PRINT USING"+####.##";A#` `+1234.56`
	`PRINT USING"+####.##";-1234.56` `-1234.56`

-	Prints a minus sign only if the expression is negative.
	`PRINT USING"+####.##";A#` `1234.56`
	`PRINT USING"+####.##";-1234.56` `-1234.56`

*	Fill the spaces before a number with asterisks. One example would be formatting ouput when printing checks. `PRINT USING"$##,###,###.##";12.34` `$********78.90`

EXAMPLE See examples on next page...

REMARK When ***error*** is printed in the format field, this indicates the occurrence of an overflow condition and replaces the number that would have been printed. An overflow condition is when the value of the expression used would have exceeded the bounderies of the format.

USING not available for string formatting. See LEFT$, RIGHT$, STRING$ and MID$.

This version allows USING without PRINT. A$=USING"####.##";232 is acceptable. See appendix for additions to exponential formatting with this version.

PRINT USING continued

FORMAT EXAMPLES

In all the examples ***A =12345.678.*** Note that .678 rounds up to .68.

PRINT USING FORMAT	RESULT
"*$###,###,###,###.##";A	**********$12,345.68
"%###.#";A/1000	%12.3
"+###,###.##";A	+12,345.68
"-###,###.##";-A	-12,345.68
"##/##/##";A	1/23/45
"##:##:##";A	1:23:45
".###,###,###,###";1.345E-8	.000,000,013,450
".############";1.345E-8	.000000013450
"###,###,###,###,###";9.123E15	9,123,000,000,000,000
"###.##E16";123E15*1E-16	12.30E16

PROGRAM EXAMPLE

```
A$="##.##"
:
PRINT USING A$;10.2,USING A$;9.237, USING A$; 4.555
PRINT 10,12,13, USING A$;12.399
:
PRINT@(0,10);USING A$;23.12321
:
PRINT%(0,295);USING "@######.##";12.33
:
OPEN"O",1,"TESTFILE"
PRINT#1, USING A$;9.999
CLOSE#1

RUN

10.20          9.24          4.56
10             12            13              12.40

23.12        <--- at text position 0,10
@12.33       <--- at graphic position 0,295
10.00        <--- To disk file "TESTFILE"
```

PSTR$ function/statements

FORMATS

function
PSTR$(*var%* **)**

statements
READ PSTR$(*var%* **)**
PSTR$(*var%* **) =** *"quoted string constant"*

DEFINITION

The statements load the address of a string constant into *var%*.

The function returns the string pointed to by *var%*.

EXAMPLE

```
DATA Andy, Dave, Scott, Mike
:
DIM D(4)
:
FOR X=1 TO 4              <---Set Pointer String to DATA items above
  READ PSTR$(D(X))
NEXT
:
"Print PSTR$ of D(n)"
FOR X=1TO4
  PRINT PSTR$(D(X))
NEXT
END
:
PSTR$(g%)="Hello"         <--- Set Pointer String to a constant
PRINT PSTR$(g%)

RUN

Andy
Dave
Scott
Mike
Hello
```

REMARK

This is a handy way to save string memory. Examples:

```
A$="Hi There!"
```
A$ will take at least 10 bytes (256 bytes if not defined). The quoted string takes another 10 bytes. Total memory used: 20 bytes

```
PSTR$(A)="Hi There!"
```
The quoted string `"Hi There!"` takes 10 bytes. The integer variable "A" takes two bytes. Total memory used: 12 Bytes

Macintosh: Use *var&* instead of *var%*.

statement **PUT**

FORMAT **PUT** (x1,y1) *variable* [(*array index* [, *array index* [,...]) [,*mode*]

DEFINITION This statement places the graphic bit image stored in an array with the GET statement, to the screen position at coordinates specified by x1,y1.

If an array has been used then you MUST specify the index number of the array (some versions of BASIC always assume an integer array. ZBasic will allow you to store bit images in any variable type as long as enought memory is available to do so.

Memory required for pixel images is calculated using this formula (based on GET (x1,y1)-(x2,y2) where x1 and y1 designate the upper right-hand-corner of the image and x2 and y2 are the pixel positions designating the lower-left-hand-corner of the image) :

$$6+((y2-y1)+1) * ((x2-x1+1) * bpp+7)/8)$$

The number of bits per pixel (bpp) depends on system colors or grey levels. See next page for specifics. Also see GET in this reference section, for detailed information about storing the pixel image in an array.

mode		
	XOR	XORs the pixels over the background pixels. This is the most usful for animation purposes and is also the default.
	OR	ORs the pixels over the existing pixels This one way to cover the background graphics (overlays the existing graphics).
	AND	ANDs the picture with the background.
	PRESET	Similar to PSET except the reverse image is shown (negative).
	PSET	Draws the image over the background exactly as created.

It is recommended that COORDINATE WINDOW be used when using GET.

EXAMPLE

```
DIM A(10000)
MODE  7                 <---- Not needed on the Macintosh version
COORDINATE WINDOW       <--- Pixel coordinates
:
CIRCLE 100,100,80
GET (0,0)-(100,100), A(1)
:
FOR x= 1 TO 200 STEP 3
  PUT (x, 90), A(1)  <--- Do it twice to XOR the pixels and move the image across
  PUT (x, 90), A(1)       the screen without disturbing the background
NEXT x
:
END
```

This routine moves a section of a circle across the screen. It is XORed to the screen twice so the item doesn't repeat and it will appear to move across the screen without disturbing the background (default PUT *mode* is XOR) .

continued...

PUT statement

REMARKS It is important to see entry under GET for more information.

Macintosh: With this version of ZBasic, PUT has another, optional, parameter: `PUT (x1,y1) [-(x2,y2)], var.` The second parameter allows you to scale the image, making it either larger or smaller by giving the rectangle size in which it is to appear. The x2, y2 parameter is the lower-right corner of the image.

Bits-per-pixel (bpp) will vary by the type of Macintosh you have. The standard black and white Macintoshes have one bit-per-pixel.

The Macintosh II may have up to 16 bits-per-pixel (with upto 256 colors or grey-levels per pixel). Check addendum of Macintosh II for specifics.

MSDOS: Bits-per-pixel (bpp) will vary with the graphics adaptor board being used:

GRAPHIC TYPE	MODE(s)	COLORS	BITS PER PIXEL (bpp)
CGA	5	4	2
CGA	7	2	1
EGA	16-19	3-16	2 (64K or less on EGA card)
EGA	16-19	16	4 (More than 64K on card)
HERCULES	20	1	1

Z80: GET and PUT are not supported with these versions of ZBasic.

Apple // ProDOS and DOS 3.3: GET and PUT are not supported with this version. See DRAW example on ProDOS disk and the BLOAD and BSAVE functions for possible alternatives.

command **QUIT**

FORMAT **QUIT**

DEFINITION QUIT is used to exit the ZBasic Standard Line editor and return control to the operating system.

EXAMPLE

```
QUIT

DOS Ready    <----DOS prompt of your system
```

REMARK We highly recommend saving your program prior to using **QUIT**.

Macintosh: You may also quit from the menu.

MSDOS: SYSTEM functions the same as QUIT.

RANDOM statement

FORMAT **RANDOM [IZE]** [*expression*]

DEFINITION Seeds the random number generator so that ZBasic produces a new sequence of random numbers.

If *expression* is used, the RND function will return a repeatable series of numbers.

EXAMPLE

```
DEFTAB 5
RANDOM 12345
FOR I = 1 TO 5
  PRINT RND(10),
NEXT  I
:
RANDOM 12345 <--- Let's see if it repeats as above.
FOR I = 1 TO 5
  PRINT RND(10),
NEXT  I: PRINT

RUN

8         1         10        4         7
8         1         10        4         7
```

```
PRINT"Press any key to set random seed"    <--- Paranoid seed routine
DO
  R=R+1
UNTIL LEN(INKEY$)
RANDOM  R
:
FOR I = 1 TO 5
  PRINT RND(10),
NEXT  I
END

RUN

Press any key to set random seed
1         8         8         5         9
```

REMARK The results of the first two passes were the same because the seed of 12345 was the same. When a different number is used, or no number, the result will be RANDOM.

If *expression* is the same, the same random pattern will be repeated with all versions of ZBasic.

The [IZE] part of RANDOM is not supported on the Apple // and Z80 versions.

statement **RATIO**

FORMAT **RATIO** *byte expression1, byte expression2*

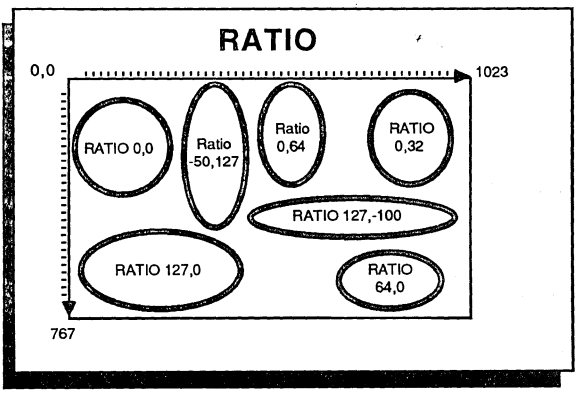
DEFINITION This statement will change the aspect ratio of graphics created with CIRCLE.

byte expression1 Horizontal ratio. A number between -128 and +127 that gives the relationship of the width of the circle to normal (zero).

byte expression2 Vertical ratio. A number between -128 and +127 that gives the relationship of the height of the circle to normal (zero).

Value	Relationship to normal		
+127	= 2.0	times normal	
+64	= 1.5	times normal	
+32	= 1.25	times normal	
0	= **0**	Normal proportion	
-32	= 0.75	times normal	
-64	= 0.5	times normal	
-96	= 0.25	times normal	
-128	= 0	times normal	(no width or height)

EXAMPLE




```
RATIO -50, 127
CIRCLE h,v,r
```

REMARK RATIO settings are executed immediately and all CIRCLE commands, including CIRCLE TO and CIRCLE PLOT will be adjusted to the last RATIO.

Also see ROUNDRECT toolbox routines for other options to creating circles with various rations.

READ# statement

FORMAT **READ #** *filenumber*, {*var* |*var*$ *; stringlength* } [, ...]

DEFINITION Reads strings or numbers saved in compressed format with WRITE# and stores them into corresponding *variables*. The list may consist of any type string or numeric variables or array variables.

filenumber	The filenumber to work from
var	Any numeric type variable
var$	String variable
; stringlength	The number of characters to load into the string variable

Important Note: A string variable **must** be followed by *;stringlength* to specify the number of characters to be read into that string.

EXAMPLE

```
REM The four variables below will require 18 bytes for storage
REM A$=4 bytes, A!= 4 bytes, A#=8 bytes, A%=2 bytes
:
A$="TEST": A!="12345.6":A#="12345.67898":A%=20000
:
OPEN"0",1, "DATAFILE", 18     <--- Write a file with a record length of 18
WRITE #1, A$;4, A!, A#, A%
CLOSE#1
:
OPEN"I" ,1,"DATAFILE", 18
READ#1, Z$;4, Z!, Z#, Z%      <---Read in same order and type (see notes)
CLOSE# 1
:
PRINT Z$, Z!, Z#, Z%
END

RUN

TEST      12345.6      12345.67898     20000
```

REMARK Note: Do not mix variable types when using READ# and WRITE#. Reading string data into numeric variables, and visa-versa, will create variables with incoherent data.

READ# and WRITE# store and retrieve numeric data in a compressed format. This saves disk space and speeds program execution.

While you may load numeric data into strings and convert using CVB or CVI, it is best to refrain from this since it requires more time and is less efficient.

See the chapter "Files" for more detailed information using random and sequential files. Also see RECORD, LOC, REC, LOF and "Disk Error Trapping".

statement **READ**

FORMAT

READ [*variable* {-or- PSTR$(*var%*) } [, . . .]]

DEFINITION

The READ statement reads strings or numbers from a DATA statement into corresponding *variables.*

The *variable* list can consist of any combination of variable types (string or numeric, including arrays).

If no variable is given the READ statement will skip one DATA item.

EXAMPLE

```
DIM P%(3)
:
DATA Joe, Smith, Harry, "@ Cost"
DATA 1234.5, 567.8, 91011.12, 1314.15
:
READ A$, B$, C$, D$     <--- Regular old fashioned READ
READ A!, B!, C!, D!
PRINT A$, B$, C$, D$
PRINT A!, B!, C!, D!
:
RESTORE                 <--- Set pointer back to start of DATA to READ again
FOR X=0 TO 3
  READ PSTR$(P%(X))     <---Use pointer string to point at DATA string constants
NEXT:PRINT
PRINT "PSTR$>";
FOR X= 0 TO 3
  PRINT PSTR$(P%(X)),
NEXT
:
RESTORE 6               <--- Set DATA pointer to the sixth item
READ A#
PRINT A#
END

RUN

Joe      Smith      Harry      @ Cost
1234.5   567.8      91011.12   1314.15

PSTR$> Joe      Smith      Harry      @ Cost

567.8
```

REMARK

Leading spaces in string data statements will be ignored unless contained in quotes.

Do not read numeric data into string variables and vice versa (no error is generated). Don't read past the end of a data list.

See RESTORE, PSTR$ and DATA.

RECORD statement

FORMAT **RECORD** [#] *filenumber*, *recordnumber* [, *location in record*]

DEFINITION The RECORD statement is used to position the file pointer anywhere in a file. Once the file pointer has been positioned you may read or write data from that position.

RECORD can position both the RECORD pointer and the location within a record.

filenumber	Filenumber from 1 to 99
recordnumber	RECORD number to point to. Default is zero.
location in record	Optional location in RECORD. Default is zero.

EXAMPLE

```
OPEN"R",1,"TESTFILE",30
:
FOR Position = 0 to 29
  RECORD #1, 6, Position
  READ#1, A$;1              <--- Reads one character at a time from record 6.
  PRINT A$;
NEXT
:
CLOSE#1
END

RUN

Fred          Stein
```

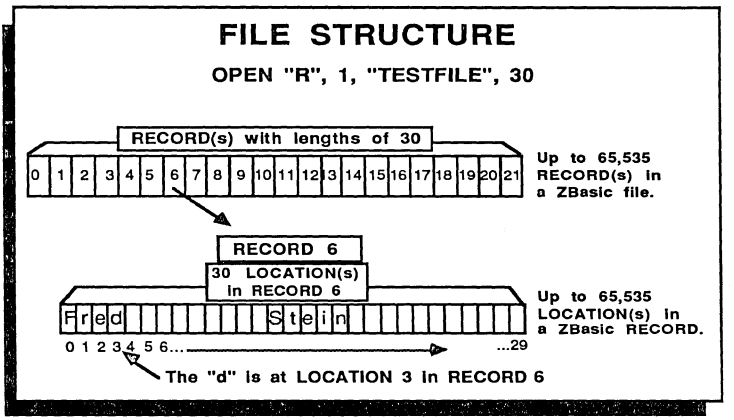
See illustration next page...

REMARK The default RECORD length is 256 bytes. The maximum record length is 65,535. The maximum number of records in a file is 65,535.

See OPEN, READ#, WRITE#, PRINT#, INPUT#, LINEINPUT#, LOC, LOF, REC, CLOSE, and the chapter entitled "Files".

The maximum record length and number of records in a file is 2,147,483,647.

RECORD continued



In the illustration, the name "Fred Stein" was stored in RECORD six of "TESTFILE".

To point to FILE #1, RECORD 6, LOCATION 3 use the syntax:

```
RECORD# 1, 6, 3
```

The location within a record is optional (zero is assumed if no location is given).

If RECORD 1, 6 had been used (without the 3), the pointer would have been positioned at the "F" in "Fred".

If RECORD is not used, reading or writing starts from the current pointer position. If a file has just been opened, the pointer is positioned at the beginning.

After each read or write, the file pointer is moved to the next position in the file.

The maximum record length and number of records in a file for this versions is 2,147,483,647.

REC function

FORMAT **REC** (*filenumber*)

DEFINITION Returns the current position of the record pointer for the file specified by *expression.* The first record in a file is record zero (0).

Also often used with REC is LOC which returns the position within the record.

EXAMPLE

```
OPEN "O",1,"THISPROG",10  <--- Record length of ten
:
A$="012345"               <--- String length of six
:
FOR I = 0 TO 3
  PRINT#1, A$;
  PRINT "On pass";I;" file position was ";
  PRINT "REC="REC(1);" and LOC=";LOC(1)
NEXT  I
:
RECORD #1, 0, 4           <--- Position the file pointer with RECORD
:
PRINT "Right after the middle RECORD statement; ";
PRINT "REC="REC(1);" and LOC=";LOC(1)
:
CLOSE#1
END
```

```
RUN

On Pass 0 file position was REC=0 and LOC=6
On Pass 1 file position was REC=1 and LOC=2
On Pass 2 file position was REC=1 and LOC=8
On Pass 3 file position was REC=2 and LOC=4

Right after the middle RECORD statement; REC=0 and LOC=4
```

REMARK The default record length is 256 bytes. LOC returns the position within a RECORD.

See OPEN, CLOSE, LOC, LOF, RECORD, READ#, WRITE# and the chapter entitled "Files".

statement **REM**

FORMAT **REM** *followed by programming remarks*

DEFINITION The REM statement is used for inserting comments or remarks into a program. ZBasic ignores everything following a REM statement.

To save time, you can type an apostrophe (') at the beginning of a line and it will be converted into a REM statement.

EXAMPLE

```
REM     This is a comment or remark
REM     ZBasic ignores everything following a REM
REM     Including any commands enbedded in the remark
:
REM     Colons are often used to make blank lines.
:
:
:
REM     Thoughful use of REM makes a program easier to read.

RUN

ZBasic Ready_
```

REMARK REM statements are not compiled and do not take up any memory in the object code.

Note: Some versions of ZBasic will not convert the apostrophe to REM.

RENAME statement

FORMAT **RENAME** *string1* {, | TO} *string2*

DEFINITION This statement is used to rename the file *string1* to the new name *string2.*

EXAMPLE

```
DIR

GOOGOO              ZBASIC.COM
FRED.BAS            OLDFILE.BAS

INPUT  "FILE NAME TO CHANGE: ";File1$
INPUT  "NEW NAME FOR FILE: ";File2$
RENAME File1$ TO File2$

RUN

FILE NAME TO CHANGE: GOOGOO
NEW NAME FOR FILE: GOONIE

DIR

GOONIE              ZBASIC.COM
FRED.BAS            OLDFILE.BAS
```

REMARK This command is also available in command mode. Remember that filename formats are different from system to system and may not be available for some machines.

TRS-80 model 1,3: RENAME not supported with these versions.

Macintosh: Pathnames or volume number may be used.
Macintosh: RENAME file1$ {TO | , } file2$ [, *volume number%*]. Also see NAME.

MSDOS: See CHDIR, PATH$, RMDIR and MKDIR in the MSDOS appendix for controlling pathnames and directories. Also see NAME.

Apple // ProDOS: Pathnames supported.

command **RENUM**

FORMAT **RENUM** [*new*] [, [*old*]] [, *increment*]

DEFINITION Used for renumbering program lines.

new	The first new assigned line number desired after renumbering is complete. default = 10
old	The first old line where you want renumbering to begin. default = 0
increment	The increment between line numbers. default = 10 (256 maximum)

If an argument is omitted the default will be used.

This command will automatically update line references (GOTO, GOSUB, etc). If a line reference is to a non-existent line, it will use the next existing line number.

EXAMPLE

```
7  IF  I = 200 THEN 567
74  PRINT I
197  I  =  I + 1: GOTO  74
567  END

RENUM
LIST

10  IF I  = 200 THAN 40
20  PRINT I
30  I  =  I + L: GOTO 10
40  END
```

REMARK Line increments are limited to 256. If you issue a RENUM command that exceeds the number of allowable lines (65,534) , an error will occur and your text will be unaltered.

If you are unsure of what the results may be, SAVE your program BEFORE renumbering!

Some versions offer options for using, or not using, line numbers with full screen editors. Check your appendix for specifics.

See RENUM*, UNNUM, INDENT and FIX in the MSDOS appendix for other options.

RESET statement

FORMAT **RESET**

DEFINITION Closes all open files and devices. Functionally identical to CLOSE without parameters.

EXAMPLE

```
OPEN"O",1,"FRED"
OPEN"I",2,"HARRY"

IF ERROR THEN RESET

END
```

REMARK See CLOSE.

Not supported on Apple // or Z80 versions of ZBasic. Simply use CLOSE without a filenumber to close all open files.

statement **RESTORE**

FORMAT **RESTORE** [*expression*]

DEFINITION This statement resets the DATA pointer to the first DATA statement or optionally to the DATA item specified by *expression.*

If the *expression* is omitted, the first DATA item is assumed. ZBasic automatically sets the pointer to the next item after each variable is READ.

EXAMPLE

```
DATA ZERO, ONE, TWO, THREE, FOUR, FIVE
DATA SIX, SEVEN, EIGHT, NINE, TEN
:
"Start"
DO
  INPUT"What item do you want";Item%
  IF (item%<0) OR (item%>10) THEN "Start"
  RESTORE Item%
  READ A$
  PRINT "Item number";Item%;" is: ";A$
UNTIL Item%=0
:
RESTORE                <--- Set to beginning of DATA
READ A$: PRINT A$
:
END

RUN

What item do you want: 4
Item number 4 is: FOUR

What item do you want: 9
Item number 4 is: NINE

What item do you want: 0
Item number 0 is: ZERO

ZERO
```

REMARK If an attempt is made to READ or RESTORE past the last DATA item, the result will be zeros or NULL strings. No error will be returned.

Also see READ, PSTR$ and DATA.

RETURN statement

FORMAT **RETURN** [*line*]

DEFINITION The RETURN statement is used to continue execution at the statement immediately following the last executed GOSUB or ON GOSUB statement.

If optional *line* is used, the last GOSUB is POPPED off the stack and a GOTO *line* is performed.

EXAMPLES

```
GOSUB "First"
:
"Second"
PRINT "RETURN comes here."
END
:
"First"
PRINT "This is a subroutine"
RETURN

RUN

This is a subroutine
Return comes here
```

```
GOSUB "Routine"
END
:
"Weird"
PRINT"Ended Here!"
STOP
:
"Routine"
PRINT"At 'Routine'"
RETURN "Weird"

RUN

At 'Routine'
Ended Here!
```

REMARK When ZBasic encounters a RETURN statement which was not called by a GOSUB, it will return to the program that executed it (either DOS or the ZBasic editor).

Using RETURN *line* WITHOUT A GOSUB or from the middle of a LONG FN will cause unpredictable (probably disastrous) system errors.

Use caution when using RETURN *line* to exit event trapping routines like DIALOG ON, MENU ON, TRON, BREAK ON...

function **RIGHT$**

FORMAT **RIGHT$**(*string* , *expression*)

DEFINITION Returns the right-most *expression* characters of *string.*

EXAMPLE

```
A$="HELLO"
:
FOR I = 0 TO 6
  PRINT  I, RIGHT$(A$,I)
NEXT  I
:
A$ = "JOHN DOE"
:
SP = INSTR(1,A$,"  ")
PRINT"LAST NAME:",
PRINT RIGHT$(A$,LEN(A$)-SP)
:
END

RUN

0
1             0
2             LO
3             LLO
4             ELLO
5             HELLO
6             HELLO
LAST NAME:    DOE
```

REMARK If expression is more than the characters available, all the characters will be returned.

See LEFT$, VAL, STR$, STRING$, SPACE$, SPC, MID$ and the chapter entitled "String Variables" in the front section of this manual.

RND function

FORMAT **RND** (*expression*)

DEFINITION The RND function returns a random integer number from 1 to *expression* .

EXAMPLE

```
RANDOM
A=9
:
FOR I=1 TO 5
  PRINT RND(A),
  PRINT RND(10000)*.0001
NEXT I
:
END
```

```
RUN

3           .9201
7           .8211
1           .0912
2           .7821
9           .0108
```

Remark Some versions of BASIC return a floating point random number between 0 and 1; use RND(10000) * .0001 to emulate this (it will slow down execution).

Also see MAYBE and RANDOM.

If the same seed number is used for RANDOM, the random numbers generated by RND will be predictable on the all versions of ZBasic.

The largest number you may use for a RND *expression* is 32,767.

statement **ROUTE**

FORMAT **ROUTE** [#] *expression*

DEFINITION This statement is used to route PRINT statements to a specified device. The following are the values to be used as *expression.*

Device number	Routes PRINT statements to
negative numbers	I/O devices; See your appendix for specifics
0	Screen (default)
1-99	Disk files specified by number
128	Printer

EXAMPLE

```
ROUTE 128
PRINT "HELLO"          <--- This HELLO goes to the printer
:
OPEN"O",1,"Test"
ROUTE 1
PRINT "HELLO"          <--- This HELLO goes to the file "Test"
CLOSE#1
:
OPEN"C",-1,300
ROUTE -1
PRINT "HELLO"          <--- This HELLO goes to a serial device
CLOSE#-1
:
ROUTE 0
PRINT"HELLO"           <--- This HELLO goes to the screen
END

RUN

HELLO
```

REMARK You should eventually route the output back to a screen device (ROUTE 0).

See PRINT, OPEN"C" and the chapter "Files" for more information.

Also see ROUTE 128, CLEAR LPRINT, DEF LPRINT and DEF PAGE for more information about routing text and graphic output to the Imagewriter and Laserwriter. Be sure to use CLEAR LPRINT with ROUTE 128 to tell the Macintosh printer driver to print the page.

RUN statement

FORMAT **RUN** [*filenumber*]

DEFINITION The RUN statement does one of two things.

RUN *filenumber* Loads a compiled chain program specified by *filenumber* and executes it:

```
OPEN"I", 1, "Prog.CHN"
RUN 1
```

RUN Clears all variables and pointers and restarts the current program from the first line.

EXAMPLE

```
OPEN"I",2,"MENU"
RUN  2                <---Loads and RUNS CHAIN program "MENU"
```

```
TRONB
FOR X=1 TO 100
  PRINT X
NEXT
RUN                   <--- RUNS this program over and over...
```

REMARK Also see the RUN command and the chapters "Running ZBasic Programs" and "Chaining" for more information.

Also see RUN *filename$, volumenumber%* in the appendix.

command **RUN**

FORMATS **RUN** [[{+ | * }]["] *filename* ["]]

DEFINITION This command is used from the Standard Line Editor to compile a program:

RUN	Compiles source code in memory and executes.
RUN *filename*	Compiles source code called *filename* from disk and executes. Source code must have been saved in tokenized format with SAVE (not as a text file).
RUN*	Compiles source code in memory and saves as a stand-alone application on disk. Asks for filename after compiling.
RUN* *filename*	Compiles source code called *filename* from disk and saves as a stand-alone application on disk. Source code must have been saved tokenized (not as a text file). Asks for filename after compiling.
RUN+	Compiles source code in memory and saves as a chain file to disk (no runtime included). Asks for filename after compiling.
RUN+*filename*	Compiles source code called *filename* from disk and saves as a chain file to disk (no runtime included). Asks for filename after compiling.

EXAMPLE

```
PRINT "THE PROGRAM RUNS!"

RUN

THE PROGRAM RUNS!
```

REMARK Compiling from disk will destroy any text currently in memory. If an error is encountered when compiling from disk , ZBasic will load the source code and print an error message.

After a successful compilation, typing MEM will return memory used for the object code and variables.

See "Executing Programs" in the front of this manual for more information about compiling large programs.

Also see COMPILE and LCOMPILE for ways of compiling a program and seeing all the compile time errors at once (instead of one at a time as with RUN).

SAVE command

FORMAT **SAVE** [[{* | +}] ["] *filename* ["]]

DEFINITION SAVE is used from the Standard Line Editor to save the source code in memory. You may save your source code in a number of formats:

SAVE	Saves program in tokenized format. This requires less room on the disk and saving and loading is much faster than with text files. In order to compile a file from disk a program must be saved in this format.
SAVE*	Saves program in TEXT or ASCII format. This allows you to load the program into other word processors or editors. Loads more slowly than SAVE above.
SAVE+	Same as SAVE* but line numbers are removed. Be sure your program doesn't uses label references with GOTO, GOSUB or other commands, since when a program is re-loaded, line numbers are added back in increments of one which will make line number references incorrect.

Note: Source code is the program you type in. Object code is the machine language program created when you compile the source code with RUN. See RUN for more information about compiling and saving compiled programs to disk.

EXAMPLE

```
SAVE* PROGRAM.TXT     <---SAVE program in ASCII (text)
SAVE  AR.BAS          <---SAVE program tokenized (condensed)
SAVE+ FILE.TXT        <---SAVE program in ASCII - with no line numbers
```

REMARK Also see LOAD, APPEND, MERGE and RUN.

statement **SELECT**

FORMAT

```
SELECT [expression or simplestring ]
   CASE [IS] relational condition [, relational condition] [,...]
      statements...
   CASE [IS] condition [, condition ][,...]
      statements...
   CASE [IS] boolean expression
      statements...
   CASE ELSE
END SELECT
```

DEFINITION

Provides a structured and efficient way of doing multiple comparisons with a single expression. While IF-THEN or LONG-IF statements could be used, they are harder to follow when reading program listings.

EXAMPLE

```
X=CARDTYPE:REM MSDOS Cardtype example.
SELECT X
  CASE 0
    PRINT"CGA CARD":MODE 7
  CASE 1
    PRINT"EGA CARD":MODE 19
  CASE 2
    PRINT"EGA with Mono": MODE 18
  CASE 3
    PRINT"HERCULES CARD":MODE 20
  CASE 255
    PRINT "Monochrome Monitor":MODE 2
  CASE ELSE
    PRINT"No Video card installed"
END SELECT
```

REMARK

See CASE and END SELECT for more examples.

Important Note: Exit a SELECT structure only at the END SELECT.

SELECT is not supported with the Apple or Z80 versions of ZBasic. Use IF-THEN or LONG-IF to accomplish the same thing.

SGN function

FORMAT **SGN** (*expression*)

DEFINITION Returns the sign of *expression*.

If *expression* is:

```
Positive            +1 is returned.
Zero                 0 is returned.
Negative            -1 is returned.
```

EXAMPLE

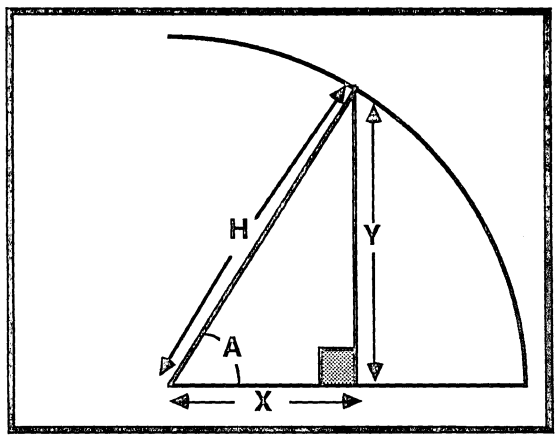
```
DEFDBL A-Z: DEFTAB 8: WIDTH 40
PRINT"       X","ABS(X)", "INT(X)","FRAC(X)",SGN(X)"
:
FOR X = -15.0 TO +15.0 STEP 3.75
  PRINT USING"-##.##";X,
  PRINT USING "##.##";ABS(X),
  PRINT USING"-##.##";INT(X),
  PRINT USING "-#.##";FRAC(X),
  PRINT USING "-#.##";SGN(X)
NEXT X

RUN

   X        ABS(X)     INT(X)     FRAC(X)     SGN(X)
-15.00     15.00     -15.00        .00       -1.00
-11.25     11.25     -11.00       -.25       -1.00
- 7.50      7.50     - 7.00       -.50       -1.00
- 3.75      3.75     - 3.00       -.75       -1.00
    .00       .00        .00        .00         .00
  3.75      3.75       3.00        .75        1.00
  7.50      7.50       7.00        .50        1.00
 11.25     11.25      11.00        .25        1.00
 15.00     15.00      15.00        .00        1.00
```

REMARK Also see UNS$, FRAC, INT, ABS and negation.

function **SIN**

FORMAT **SIN** (*expression*)

DEFINITION The **SIN** function returns the sine of the *expression* in radians.



SIN(A)=Y/H, H*SIN(A)=Y, Y/SIN(A)=H

EXAMPLE

```
X#=SIN(123)
PRINT SIN(X2#)
```

REMARK SIN is a scientific function. The precision for scientific functions may be configured. See "Configure" in the front of this manual for more information.

See the "Math" and "Expressions" sections of this manual and ATN, TAN, COS, EXP, SQR, ^.

INTEGER SINE: ZBasic provides a predefined USR function to do hi-speed integer sines. This speeds up sine speed by up to 30 times:

USR8(*angle*) returns the integer sine of *angle* in the range ±255 (corresponding to ±1). The *angle* must be in brads. See CIRCLE for examples of brads. Example:

```
MODE7 :CLS
FOR I=0 TO 255
  PLOT I<<2,-USR8(I)+384
NEXT I
```

SOUND statement

FORMAT **SOUND** *frequency, duration*

DEFINITION **SOUND** may be used to create sound effects or music.

frequency Frequency 120 Hz to 10,000 Hz.
duration Duration in 1 millisecond increments.

Note: Hz (Hertz) represents cycles-per-second.

EXAMPLE

```
DO
  INPUT"Tone: ";Tone
  INPUT"Duration: ";Duration
  :
  SOUND Tone, Duration
  :
UNTIL (Tone=0) OR (Duration=0)
```

Example frequencies you may use in your program to create music or sound effects. (Choose the duration as required.) Quality of sound may vary by machine.

	OCTAVES						
NOTES	1	2	3	4	5	6	7
C	33	66	132	264	528	1056	2112
C^b	35	70	140	281	563	1126	2253
D	37	74	148	297	594	1188	2376
E^b	39	79	158	316	633	1267	2534
E	41	82	165	330	660	1320	2640
F	44	88	176	352	704	1408	2816
G^b	46	93	187	375	751	1502	3004
G	49	99	198	396	792	1584	3168
A^b	52	105	211	422	844	1689	3379
A	55	110	220	440	880	1760	3520
B^b	57	115	231	462	924	1848	3696
B	61	123	247	495	990	1980	3960

REMARK Some computers may not have sound. See your computer appendix for more information.

CP/M-80: Sound not supported. CHR$(7) may sound a bell on some sytems.
TRS-80 model 1,3: Requires that a speaker be connected to the cassette port.
TRS-80 Model 4: Frequency range of internal speaker limited to 0,0 to 7,31.

See appendix for using four voice sound and utilizing the sound buffer.

function **SPACE$**

FORMAT **SPACE$** (*expression*)

DEFINITION Returns a string of spaces *expression* characters long (range of 0 to 255).

EXAMPLE

```
PRINT "ZEDCORZEDCORZE"
FOR X=7 TO 0 STEP -1
  PRINT SPACE$(X);"ZEDCOR"
NEXT
PRINT"ZEDCORZEDCORZEDCOR"
END

RUN

ZEDCORZEDCORZE
       ZEDCOR
      ZEDCOR
     ZEDCOR
    ZEDCOR
   ZEDCOR
  ZEDCOR
 ZEDCOR
ZEDCOR
ZEDCORZEDCORZEDCOR
```

REMARK See STRING$, MID$, RIGHT$, LEFT$, INSTR and SPC.

SPC function

FUNCTION **SPC** (*expression*)

DEFINITION SPC prints *expression* spaces from 0 to 255.

Prints the number of spaces specified by *expression*.

EXAMPLE

```
DO
   PRINT"*";SPC(RND(20));"+"
UNTIL LEN(INKEY$)
```

RUN

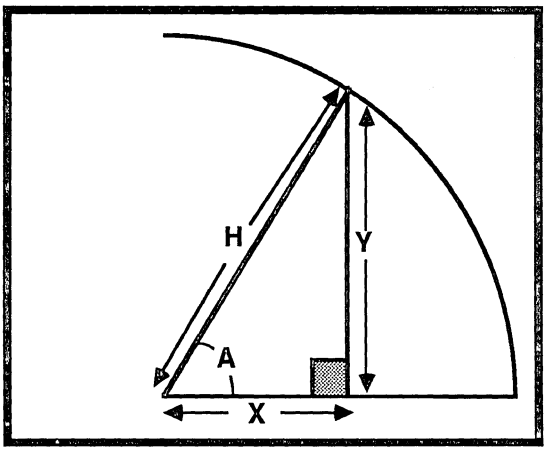
```
                    +
                   +
  +
         +
                        +
+
```

REMARK Also see SPACE$, LEFT$, STRING$, RIGHT$, MID$ and INSTR.

function **SQR**

FORMAT **SQR** (*expression*)

DEFINITION The SQR function returns the square root of *expression* .



H=SQR(X*X+Y*Y)

EXAMPLE

```
A=9
PRINT SQR(A)
```

RUN

```
3
```

REMARK SQR is a scientific function. Scientific functions may be configured to a different precision. See "Configure" in the front of this manual for more information.

For more information on scientific functions see the "Math" and "Expression" sections of this manual and ATN, SIN, COS, TAN, EXP and ^ .

STEP statement

FORMAT

FOR *variable* = *expr1* TO *expr2* [**STEP** *expr3*]
"
"
"
NEXT [*variable*] [,*variable*. . .]

DEFINITION

This parameter allows you to set the increments used in a FOR-NEXT loop. If STEP is omitted than one is assumed.

EXAMPLE

```
FOR X= 0 TO 10 STEP 2
  PRINT X;
NEXT
:
FOR X = 10 TO 0 STEP -1
  PRINT X;
NEXT
END

RUN

0 2 4 6 8 10
10 9 8 7 6 5 4 3 2 1 0
```

REMARK

Also see FOR, NEXT, DO, UNTIL, WHILE, WEND and the chapter on "Loops".

IF STEP =0 an endless loop will result.

If *expr1* or *expr3* change while the loop is executed this change will be in effect when NEXT is encountered.

Avoid long or complex loop expressions for *expr1* or *expr3* as they are evaluated every loop and will slow execution.

statement **STOP**

FORMAT **STOP**

DEFINITION **STOP** halts execution of a ZBasic program and prints the line number where execution stopped (if line numbers weren't used the lines are numbered in increments of one).

STOP when used from ZBasic will return to the Standard Line Editor.

STOP when used from a stand-alone program will return to the operating system.

EXAMPLE

```
PRINT"HELLO"
STOP

RUN

Break in 00002
ZBasic Ready
```

REMARK STOP closes all files.

END may be used when no message is desired.

See TRONB and TRONX for ways of inserting break points in your programs so that <BREAK> may be used to exit a running a program.

STR$ function

FORMAT STR$ (*expression*)

DEFINITION STR$ returns the string equivalent of the number represented by *expression*. This is used to convert numbers or numeric variables to a string.

This function is the compliment of VAL. VAL returns the numeric value contained in a string.

EXAMPLE

```
Integer% =20000
Single! =232.123
Double# = .12323295342
:
A$=STR$(Integer%)   :PRINT A$
A$=STR$(Single!)    :PRINT A$
A$=STR$(Double#)    :PRINT A$
:
X#=VAL(A$)
PRINT X#

RUN

 20000
 232.123
 .12323295342
 .12323295342
```

REMARK Also see BIN$, OCT$, HEX$, MKI$, CVI, MKB$, CVB and VAL.

function **STRING$**

FORMAT

**STRING$** (*expr*$_1$, *string*)

STRING$ (*expr1* , *expr2*)

DEFINITION

Returns a string of the length *expr1* consisting of the characters specified by either the ASCII equivalent of *expr2* or the first character of *string* .

EXAMPLE

```
PRINT STRING$ (5,"#")
PRINT STRING$ (10,65)
PRINT STRING$ (10,CHR$(65))
:
A$ = STRING$(3,"*") + "TEST"+ STRING$(3,"&")
PRINT A$
END

RUN

#####
AAAAAAAAAA
AAAAAAAAAA
***TEST&&&
```

REMARK

STRING$ is more efficient than using an equivalent string of characters.

See SPACE$, LEFT$, RIGHT$, MID$, INSTR, VAL, STR$, INDEX$ and SPC.

SYSTEM statement

FORMAT **SYSTEM**

DEFINITION Same as END. Provided for compatibility with other versions of BASIC.

EXAMPLE

```
PRINT"HELLO"
SYSTEM

RUN

HELLO
```

REMARK Functionally identical to the ZBasic END statement. See END and STOP.

Not Supported with Apple // or Z80 versions of ZBasic. Use END.

statement **SWAP**

FORMAT

SWAP *var1* , *var2*

DEFINITION

SWAP exchanges the contents of *var1* and *var2* . The variables can be of any type except INDEX$ variables.

Var1 and *var2* must be of the same type.

EXAMPLE

```
B$="YES"
A$="NO"
PRINT A$, B$
SWAP A$, B$
PRINT A$, B$
PRINT
:
A=1:B=100
PRINT A,B
SWAP A,B
PRINT A,B
END
```

```
RUN

YES             NO
NO              YES

1                100
100              1
```

REMARK

SWAP will execute faster and take less memory than similar methods using "holding variables".

SWAP *does not* function with INDEX$.

TAB function

FORMAT **TAB** (*expression*)

DEFINITION Tab will move the cursor to the positions; 0 through 255, designated by *expression.*

Three devices may be used with Tab:

DEVICE	FORM	WILL POSITION
SCREEN	PRINT	CURSOR
PRINTER	LPRINT	PRINT HEAD
DISK	PRINT#	FILE POINTER

EXAMPLE

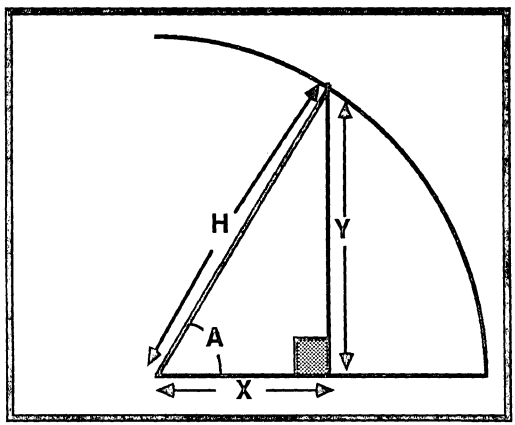
```
DATA Fred Smith, 12 E. First, Tucson, AZ, 85712
DATA Dana Andrews, 32 Main, LA, CA, 90231
:
PRINT "Name"TAB(15)"Address"TAB(30)"City"TAB(40)"State  ZIP"
PRINT STRING$(50,"-")
:
FOR Item= 0 TO 1
  RESTORE Item*5
  READ N$, A$, C$, S$, Z$
  PRINT N$ TAB(15) A$ TAB(30) C$ TAB(40) S$"     "Z$
NEXT
END

RUN

Name           Address        City      State  ZIP
--------------------------------------------------
Fred Smith     12 E. First    Tucson    AZ     85712
Dana Andrews   32 Main        LA        CA     90231
```

REMARK Tab will start numbering from the zero position. Also see DEFTAB, PRINT@, PRINT%, POS, PAGE, WIDTH and WIDTH LPRINT.

function **TAN**

FORMAT **TAN** (*expression*)

DEFINITION Returns the value of the tangent of the *expression* in radians.



TAN(A)=Y/X, X=Y/TAN(A), Y=TAN(A)*X

EXAMPLE

```
X#=TAN(T+Z)/3
```

REMARK TAN is a scientific function. Scientific accuracy may be configured differently than single or double precision. See "Configure" at the beginning of this manual.

Also see ATN, COS, SIN, EXP, SQR and ^.

For more information on scientific functions see "Math" and "Expressions" in the front section of this manual.

TIME$ function

FORMAT **TIME$**

DEFINITION Returns an eight character string which represents the systems clock value in the format HH:MM:SS where HH=1 to 24 hours, MM= 0 to 60 minutes, SS= 0 to 60 seconds.

EXAMPLE

```
PRINT TIME$
DELAY 1000
A$=TIME$
PRINT A$

RUN

10:23:32
10:23:33
```

REMARK See DATE$ and DELAY.

This function will return a 00:00:00 if the system or version has no clock.

MacIntosh: Set time from the Control Panel Desk Accessory. Also see TIMER for other ways of getting seconds.

MSDOS: Set time using TIME$= hh, mm, ss. Also see TIMER.

Apple: See appendix for variations of system clocks.

Z80: See appendix for your particular hardware.

statement **TROFF**

FORMAT **TROFF**

DEFINITION TROFF is used to turn off the trace statements: TRON, TRONX, TRON and TRONS.

EXAMPLE

```
TRON
FOR X=1 TO 3
NEXT
:
TROFF
PRINT "Line tracing now off"
:
FOR X=1 TO 10
NEXT

RUN

00001 00002 00003 00002 00003 00002 00003 00004 Line tracing
now off
```

REMARK See also TRON, TRONS, TRONB, TRONX and the chapter on "Debugging Tools".

TRON statement

FORMAT **TRON** [{ B | S | X}]

DEFINITION These statements are used for tracing program execution, single stepping through a program, and setting break points for monitoring the <BREAK> key so that you can break out of a program.

TRACING PROGRAM FLOW

TRON	Prints the line numbers of the program as each line is executed so you can trace program flow and check for errors.
TRON S	Lets you single step through a program. Program execution will pause at the beginning of every line in the program following TRON S (up to the end of the program or when a TROFF is encountered). Press any key to continue or press the <CTRLZ> key to enable/disable single-stepping. <BREAK> also works.

SETTING BREAK POINTS

TRON X	Sets a break point at that line in a program and checks to see if the <BREAK> key has been pressed.
TRON B	Sets a break point at the beginning of every line in the program following it (up to the END or until a TROFF is encountered).

Note: The <BREAK> key is checked at the beginning of a line. IF <BREAK> is encountered in a program compiled with RUN, program exits to the Standard Line Editor. If <BREAK> is encountered in a stand-alone program, exit is to the system.

<CTRL S> will pause execution when encountered during execution of TRONB, TRONX or TRON. Any key will restart. <CTRL Z> will activate/deactivate single-step mode when any TRON is active. Note: INKEY$ may lose keys if TRON is used.

EXAMPLE

```
TRON:TRONS
:
PRINT "HELLO"
:
TROFF

RUN

00001 <KEY> 00002 <KEY> 00003 <KEY> HELLO 00004 <KEY>
```

REMARK Every line between a TRON and TROFF may use up to eight extra bytes per line. Use TRON sparingly to save memory and increase execution speed. See chapter entitled "Debugging Tools" for more information. INKEY$ may lose keys with TRON.

Macintosh: <BREAK> is <Command Period>. Also see BREAK ON, and TRON WINDOW in appendix for other ways of tracing program flow and variable values.
MSDOS: <BREAK> is <CTRL C>.
CP/M: <BREAK> is <CTRL C>.
Apple // ProDOS or DOS 3.3: <BREAK> is <CTRL C> or <CTRL RESET>.
TRS-80: <BREAK> is <BREAK>.

function **UCASE$**

FORMAT **UCASE$** (*string*)

DEFINITION Returns a string with all characters converted to uppercase (capital letters).

EXAMPLE

```
PRINT UCASE$("hello")
:
A$="HeLLo"
PRINT UCASE$(A$)
END

RUN

HELLO
HELLO
```

```
DO
  key$=UCASE$(INKEY$)
UNTIL LEN (key$)
PRINT key$
END

RUN

S           <---always returns an uppercase character
```

```
REM This function converts a string to Lowercase
:
LONG FN lcase$(string$)
  FOR X=1 TO LEN(string$)
    A=PEEK(VARPTR(string$)+X)
    IF (A>64) AND (A<91) THEN A=A+32
    POKEVARPTR(string$)+X,A
  NEXT
END FN=string$
:
PRINT FN lcase$("HELLO")

RUN

hello
```

REMARK This function is very useful when sorting data containing upper and lower case and for checking user input without regard to case.

Also see LEFT$, RIGHT$, MID$, INSTR, STR$, VAL, and the chapter "String Variables" in this manual.

UNS$ function

FORMAT **UNS$** (*expression*)

DEFINITION Returns a string which equals the integer value of *expression* in an unsigned decimal format.

EXAMPLE

```
PRINT UNS$(-1)
PRINT UNS$ (4)
:
PRINT
PRINT 65535

RUN

65535
00004

-1
```

REMARK This function is useful for displaying integers in an unsigned format (0 through 65,535 instead of -32,768 through 32,767).

See STR$, DEC$, OCT$, HEX$, VAL and the chapter on "Numeric Conversions".

See DEFSTR LONG for enabling this function to work with LongIntegers.

statement **UNTIL**

FORMAT

DO
.
.
UNTIL *expression*

DEFINITION

UNTIL is used to mark the end of a DO loop.

The DO loop repeats until the *expression* following the UNTIL is true (non-zero).

A DO loop will always execute at least once.

EXAMPLE

```
DO
  X=X+1
UNTIL X=100
PRINT X
:
"Wait for a key"
DO
  I$=INKEY$
UNTIL LEN(I$)
END

RUN

100
<KEY PRESS>
```

REMARK

Notice ZBasic will automatically indent DO loop structures two spaces. See the chapter on "Formatting Program Listings" for other ways of formatting listings.

Also see FOR, NEXT, STEP, WHILE, WEND and the chapter on "Loops" in the technical section of the manual.

WHILE,WEND may be used to exit a loop immediately if a condition is false.

USR function

FORMAT **USR** *digit* (*word expression*)

DEFINITION The USR function calls the user created subroutine, defined with DEFUSR, specified by a *digit* 0 to 9, and returns the value of integer *expression* in the 16 bit accumulator.

EXAMPLE

```
REM EXAMPLE ONLY     DO NOT USE!
:
DEFUSR2 = LINE "Routine two"
X=USR2(938)
PRINT X
END
:
"Routine two"
MACHLG  &8B,&C4,&C3:RETURN

RUN

23921
```

REMARK A machine language return is necessary at the end of a USR routine.

ZBasic provides pre-defined USR functions that perform some powerful functions like integer sine and cosine. See next page.

Macintosh: Be sure to use LongIntegers whenever referencing memory addresses. Also see CALL in the Macintosh appendix.

MSDOS: See CALL in your appendix.

Apple ProDOS: See MLI in the ProDOS appendix.

Predefined USR functions

These pre-defined USR functions are available for all versions of ZBasic. See your Computer Appendix for possible other USR functions.

USR6(*expr*)
Returns the last line number executed that used any of the TRON functions (*expr* is not used).

```
TRONX
I=USR6(0)
PRINT I
```

USR7(*expr*)
Returns ZBasic's random number seed used in the RND function (*expr* is not used).

```
FOR I=1 TO 10
  PRINT USR7(0)
NEXT I
```

USR8(*angle*)
Returns the integer sine of *angle* in the range ±255 (corresponding to ±1). The angle must be in brads.

```
MODE7 :CLS
FOR I=0 TO 255
  PLOT I<<2,-USR8(I)+384
NEXT I
```

USR9(*angle*)
Returns the integer cosine of *angle* in the range ±255 (corresponding to ±1). The angle must be in brads.

```
MODE7 :CLS
FOR I=0 TO 255
  PLOT I<<2,-USR9(I)+384
NEXT I
```

USR statement

FORMAT **USR** *digit* (*expression*)

DEFINITION This statement will call the USR routine defined by DEFUSR*digit* and transfer the result of *expression* in the integer accumulator.

EXAMPLE

```
Example only DO NOT USE
:
DEFUSR0=LINE "Machine language"
USR0(0)
END
:
"Machine Language"
MACHLG &39, &C9:  RETURN
```

REMARK The USR routine must be set by the program or be a predefined USR routine. Also see DEFUSR, USR function, LINE, CALL, MACHLG, the chapter about "Machine Language" in this manual, and your computer appendix.

Macintosh: Be sure to use LongIntegers whenever referencing memory addresses. Also see CALL in the MAcintosh appendix.

MSDOS: See CALL in your appendix.

Apple ProDOS: See MLI in the ProDOS appendix.

function **VAL**

FORMAT **VAL** (*string*)

DEFINITION Returns the numeric numeric value of the first number in a string.

The VAL function will terminate conversion at the first non-numeric character in *string* .

This function is the compliment of STR$. STR$ will convert a numeric expression to a string.

EXAMPLE

```
A$="HELLO"
B$="1234.56
C$="99999"
:
PRINT "The value of A$=";VAL(A$)
PRINT "The value of B$=";VAL(B$)
PRINT "The value of C$=";VAL(C$)
:
PRINT
PRINT "The value of 9876.543=";VAL("9876.543")
END

RUN

The value of A$= 0
The value of B$= 1234.56
The value of C$= 99999

The value of 9876.543= 9876.543
```

REMARK The numeric value returned by VAL will be in floating point format

See STR$, UNS$, HEX$, OCT$ and BIN$, INT, FRAC, ABS, FIX.

Also see the chapter on "Math" and "Expressions" in the front section of this manual.

VARPTR function

FORMAT **VARPTR** (*variable*)

DEFINITION Returns the address of a *variable* . Any variable type may be used except INDEX$.

EXAMPLE

```
A$="HELLO"
:
PRINT "Address of A$=";VARPTR(A$)
PRINT "Length of A$ =";PEEK(VARPTR(A$))
:
PRINT "Contents of A$= ";
FOR X=1 TO LEN(A$)
  PRINT CHR$(PEEK(VARPTR(A$)+X));
NEXT
END

RUN

Address of A$= 23456
Length of A$ = 5
Content of A$= HELLO
```

REMARK The following paragraphs describe which address VARPTR will be pointing to with different variable types.

INTEGER	Points to the 1st byte of an integer variable
SNG/DBL	Points to the sign/exponent byte
STRING	Points to the length byte
ARRAY	Points to the element specified

See the sections in the front of this manaul for the variable type you interested in to see how variables are stored in memory.

Macintosh: Be sure to use LongIntegers to store addresses.

MSDOS: var=VARPTR(*var*) returns two values: The address of *var* and the segment of *var* in a special variable called VARSEG. See appendix for details.

statement **WEND**

FORMAT

WHILE *expression*
.
.
WEND

DEFINITION

This statement is used to terminate a WHILE loop. When *expression* becomes false the loop will exit at the first statement following the WEND.

EXAMPLE

```
"Get a YES Answer and nothing else!"
INPUT"What is your answer <Y/N>:";A$
WHILE A$ <>"Y"
  INPUT"Please reconsider and say <Y>:";A$
WEND
PRINT"Thank you for seeing things my way..."
:
program continues....
```

RUN

```
What is your answer <Y/N>: N
Please reconsider and say <Y>: Y
Thank you for seeing things my way...
```

```
WHILE X*X <23000
  PRINT X*X,
  X=X+1
WEND
END
```

RUN

```
0       1       4       9       16...
```

REMARK

ZBasic will automatically indent all lines two spaces between WHILE and WEND when you use LIST. This makes programs much easier to read.

Also see FOR, NEXT, STEP, DO, UNTIL and the chapters on "Loops" and "Structure" in the front of this manual.

A structure error will occur if a WHILE exists without a matching WEND. To find a missing WEND, LIST the program and track back from the last indent.

WHILE statement

FORMAT

WHILE *expression*
.
.
WEND

DEFINITION

In a WHILE statement, *expression* is tested for true before the loop is executed and will exit to the statement immediately following the matching WEND when *expression* becomes false.

EXAMPLE

```
"GET A KEY"
WHILE LEN(Key$)=0
  Key$=INKEY$
WEND
PRINT Key$
END

RUN

<key pressed>
```

```
WHILE X<100
  X=X+1
WEND
PRINT X
END

RUN

100
```

REMARK

ZBasic will automatically indent all lines two spaces between the WHILE and WEND when you use LIST. This makes programs much easier to read.

Also see FOR, NEXT, STEP, DO, UNTIL and the chapters on "Loops" and "Structure" in the front of this manual.

A structure error will occur if a WHILE exists without a matching WEND. To find a missing WEND, LIST the program and track back from the last indent.

statement **WIDTH**

FORMAT **WIDTH [LPRINT] [=]** *byte expression*

DEFINITION Sets the allowable number of characters on a line before generating an automatic linefeed.

Theoptional LPRINT designates printer width.

If *byte expression* is set to 0, ZBasic will not send an automatic CR/LF. The range of *byte expression* is 0 to 255.

EXAMPLE

```
10 X=X+1
20 PRINT X
30 GOTO 10

WIDTH  8
LIST

00010 X=
      X+
      1
00020 PR
      IN
      T
      X
00030 GO
      TO
       1
      0
```

REMARK The default setting for the screen width is zero which disables the auto CR/LF after the limit has been reached.

To return WIDTH to normal, type WIDTH 79 (for 80 column screens) or WIDTH 0. When widths are set, listings are wrapped around nicely for easy reading.

To effect a smaller width, set byte expression to the width desired. To assure valid results for the POS statement and to keep the line position count used by tabs correct, be sure WIDTH is set to the actual screen width minus one.

WRITE# statement

FORMAT **WRITE#***expr1*, {*var %*} | *var!* | *var#* | {var$;*stringlength* } [, ...]

DEFINITION Writes the contents of string or numeric variables in compressed format to a disk file (or other device) specified by *expr1*. The list may consist of any variable type or types, string or numeric, including arrays, in any order. *Constants may not be used!*

A string variable ***must*** be followed by *;stringlength* which specifies the number of characters of that string to be written.

If the string is longer than *stringlength,* only those characters in range will be written. If the string is shorter than *stringlength,* the extra characters will be spaces.

READ# is the statement normally used to read back data written with WRITE# and will automatically read back the data written in compressed format.

EXAMPLE

```
REM The four variables below will require 18 bytes for storage
REM A$=4 bytes, A!= 4 bytes, A#=8 bytes, A%=2 bytes
:
A$="TEST": A!="12345.6":A#="12345.67898":A%=20000
:
OPEN"0",1, "DATAFILE", 18     <--- Write a file with a record length of 18
WRITE #1, A$;4, A!, A#, A%
CLOSE#1
:
OPEN"I" ,1,"DATAFILE", 18
READ#1, Z$;4, Z!, Z#, Z%      <---Read in same order and type (see notes)
CLOSE# 1
:
PRINT Z$, Z!, Z#, Z%
END

RUN

TEST      12345.6      12345.67898     20000
```

REMARK Note: Do not mix variable types when using READ# and WRITE#. READ# and WRITE# store and retrieve numeric data in a compressed format. This saves disk space and speeds program execution.

See the chapter "Files" for more detailed information using random and sequential files. Also see RECORD, LOC, REC, LOF and "Disk Error Trapping".

continued...

WRITE# continued

READ# AND WRITE# STRINGS WITH VARIABLE LENGTHS

READ# and WRITE# offer some benefits over PRINT# and INPUT# in that they will read and write strings with **ANY** embedded ASCII or BINARY characters.

This includes quotes, commas, carriage returns, control codes or any ASCII characters in the range of 0-255.

The following programs demonstrate how to save strings in condensed format, using only the amount of storage required for each string variable.

WRITE#

```
OPEN"O",1,"NAMES"
REM LB$=LENGTH BYTE
DO
  INPUT"Name: "; N$
  INPUT"Age:"; AGE
  LB$=CHR$(LEN(NAME$))
  WRITE#1,LB$;1,N$;ASC(LB$),AGE
UNTIL N$="END"
CLOSE#1
END
```

READ#

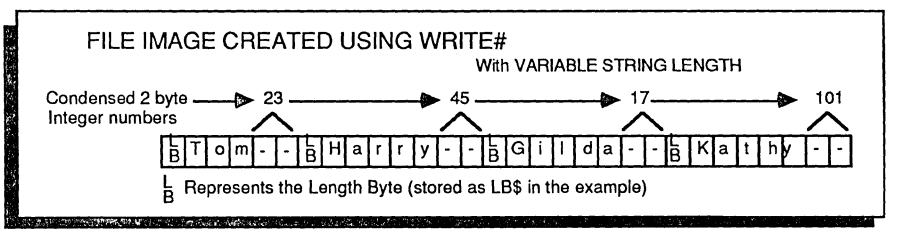
```
OPEN"I",1,"NAMES"
REM LB$=LENGTH BYTE
:
DO
  READ#1, LB$;1, N$;ASC(LB$), AGE
  PRINT N$","AGE
UNTIL N$="END"
CLOSE#1
END
```

The WRITE# program stores a one byte string called LB$ (length byte). The ASCII of this string (a number from 0 to 255) tells us the length of N$.

Notice in line 4 of READ#, that LB$ is read BEFORE N$, thus allowing us to read the length of N$ first. All data in file handling statements is processed IN-ORDER.

This illustration shows how the data is saved to the disk when string data is saved using the variable length method. LB for "Tom" would be 3, LB for "Harry" would be 5, etc.

VARIABLE STRING LENGTH WRITE#



XELSE statement

FORMAT

LONGIF *expression*
.
XELSE
.
ENDIF

DEFINITION

This statement is used to separate the FALSE from the TRUE section of a LONG IF structure.

The statements following the XELSE will only be executed if the statement following the LONG IF is false.

EXAMPLE

```
LONGIF  10 = 0
  PRINT"TRUE"
XELSE
  PRINT"FALSE"
ENDIF
END

RUN

FALSE
```

REMARK

All program lines between the LONG IF and XELSE are indented two characters when using LIST. This makes a program easier to read.

A structure error will occur the XELSE does not have a matching LONG IF.

operator **XOR**

FORMAT

expression$_1$ **XOR** *expression*$_2$

DEFINITION

Provides a means of doing a logical EXCLUSIVE OR on two *expressions* for IF-THEN testing and BINARY operations.

This operator will return true if one condition is true and one condition is false. False will be returned if both conditions are true or both conditions are false.

EXAMPLE

```
A$="Hello"
IF A$="Hello" XOR A$="Goodbye" PRINT "YES"
IF A$="Hello" XOR A$="Hello" PRINT "YES"

RUN

YES
```

REMARK

XOR TRUTH TABLES

condition XOR condition TRUE(-1) if only one condition is TRUE, else FALSE(Ø)

XOR	BOOLEAN "16 BIT" LOGIC		
1 XOR 1 = Ø	ØØØØØØØ1		1ØØØØ1Ø1
Ø XOR 1 = 1	XOR ØØØØ1111	XOR	1ØØØØ111
1 XOR Ø = 1	= ØØØØ1111Ø	=	ØØØØØØ1Ø
Ø XOR Ø = Ø			

```
FALSE XOR FALSE = FALSE
TRUE  XOR FALSE = TRUE
FALSE XOR TRUE  = TRUE
TRUE  XOR TRUE  = FALSE
```

MSDOS APPENDIX

MSDOS APPENDIX

ZBasic MSDOS™
Version 4.0

For MSDOS™, PCDOS™
and IBM® PC and compatible computers

Original code by
Scott Terry

4.0 Enhancements by
Halbert P. Liang

ZEDCOR, Inc.

MSDOS APPENDIX

TABLE OF CONTENTS

MSDOS APPENDIX

TABLE OF CONTENTS

MSDOS APPENDIX

GETTING STARTED

This version of ZBasic is provided on a 180K diskette to be compatible with most MSDOS computers. It is recommended that you make a backup of the diskette and put the master away for safekeeping.

HOW TO MAKE A BACKUP

1. Put an MSDOS or PCDOS diskette in drive A: and start the system.
2. Put a Write protect tab on the ZBasic master diskette.
3. Put the ZBasic disk in drive A: and a blank diskette in drive B:
4. Type: `DISKCOPY A: B:`
5. Put the master ZBasic diskette away for safekeeping.

HOW TO PUT ZBASIC ON A BOOTABLE MSDOS DISKETTE

1. Put an MSDOS diskette in drive A: and start the system
2. Put a blank diskette in drive B:
3. Type: `FORMAT/S B:`
4. Take the newly created diskette out of B: and put it in drive A:
5. Put the ZBasic diskette in B:
6. Type: `COPY B:*.* A:`

HOW TO COPY THE ZBASIC FILES TO A HARD DISK:

1. Put the ZBasic disk in drive A:
2. Type: `Copy A:*.* C:`

NECESSARY FILES

The only necessary file for running ZBasic is: `ZBasic.COM`.
If you are using a Hercules board then `ZHERC.BAT and HERC.COM` are required when mixing text and graphics. If only graphics are used these files are not necessary (these files are not required with other graphic boards). All the other files are optional. See "Files included on the Master Diskette" on the next page.

RUNNING ZBASIC

To run ZBasic from DOS just type: `ZBASIC`. If you are using a Hercules board type; `ZHERC`. For more detailed information see "Running ZBasic" in the main section of this manual and "Executing ZBasic from MSDOS" in this appendix.

TWO EDITORS

You will see a start-up screen as described in "Getting Started" in the front section of this manual. Press "E" to go into the Standard Line Editor. Use the <F10> key to toggle between the Standard Line Editor (where all direct commands are executed) and the Full Screen Editor.

CONFIGURING ZBASIC FOR YOUR COMPUTER

You may configure many of the default settings of ZBasic to your preferences and for your computer type. See "Configure" in the front section of this manual and "MSDOS Specific Configuration Options" in this appendix for ways of changing the standard settings.

MSDOS APPENDIX

ENHANCEMENTS TO 3.02

When ZBasic was first introduced for MSDOS machines in late 1985 we knew it was a great product, but we were also realistic and knew that there must have been things we overlooked and that people would want. So we listened carefully to all the feedback we received. This version of ZBasic is the culmination of that feedback. We have worked hard to add the features you wanted (let us know what else you would like).

The following enhancements have been made to ZBasic 3.0:

UNLIMITED VARIABLE SIZE: for String and BCD arrays. Array sizes may be up to available memory (640K on most systems). In addition, Integer variables and INDEX$ may be up to 64K (since data storage may be so large we had to change the way that VARPTR worked. See VARPTR in this appendix for specifics).

EGA GRAPHICS: New modes have been added to take advantage of the Enhanced Graphics Adapter. Resolution up to 640x350 in 16 colors is available. MODE 16-19 are used for EGA modes. CGA modes may also be emulated. See MODE, SCREEN, COLOR and PALETTE for details.

HERCULES AND HERCULES PLUS GRAPHICS: The popular "HERC" board is now supported in MODE 20. Resolution is up to 720x348. See MODE and "Hercules Graphics" in this appendix for details.

LOTS OF NEW COMMANDS: We have made ZBasic even more compatible with other BASIC languages. Some of the commands added since the last version:

COORDINATE	COORDINATE WINDOW	GET(graphics)	PUT
SCREEN	CARDTYPE	BEEP	BLOAD
BSAVE	DEF SEG	SHELL	CONFIG
CHDIR	COM ON	COM OFF	COMMAND$
CSRLN	EOF	MKDIR	PATH$
RMDIR	VARSEG	KEY ON	KEY OFF
KEY LIST	COM BUFF	WAIT	PALETTE
END SELECT	CASE	SELECT CASE	CASE ELSE

FULL SCREEN EDITOR: We have included a powerful and easy to full screen editor with ZBasic. To toggle between the Line Editor and the Full Screen Editor just press F10. Instructions are at the back of this appendix.

IMPROVED USER INTERFACE: Both editors display function key equivalents at the bottom of the screen. Just press the appropriate function key to do the command. LOAD and SAVE automatically append the pathname and .BAS suffix where appropriate. If you use another suffix just add it under "Configure".

ADVANCED COMMUNICATION FEATURES: An interrupt driven, 32K buffer has been added that supports both communication ports. See COM ON, COM OFF, COM BUFF and the chapter in this appendix entitled "RS-232 Communications".

NEW CONFIGURATION OPTIONS: We have added options to configure "Spaces between Keywords" so that you can embed keywords in variables and an option to set expression evaluation to do floating point like QuickBASIC and BASICA. See "Converting QuickBASIC and BASICA Programs" in this appendix and "Converting Old Programs" in the main reference section.

MSDOS APPENDIX

FILES INCLUDED ON THE MASTER DISKETTE

SYSTEM FILES

	FILENAME	DESCRIPTION
ZBASIC	ZBASIC.COM	This is the main ZBasic compiler and editor. Just type ZBASIC from the MSDOS to execute.
DEMO	ZDEMO.COM	A limited demo version of ZBasic (public domain). Feel free to give it away to your friends, relatives and co-workers. This and ZBASIC.HLP may be given away together (please do not give away any other programs on this disk).
HERCULES GRAPHICS	ZHERC.BAT	This batch file loads the Hercules text driver and ZBasic into memory in the correct order. Always use it when with Hercules Graphics boards.
	HERC.COM	This file is necessary for intermixing text and graphics in Hercules graphics modes. If you are creating applications that may be run on a Hercules Graphics adaptor be sure to have this file available for the program.

EXAMPLE FILES

	FILENAME	DESCRIPTION
HELP	ZBASIC.HLP	This is the help file. It is not necessary for using ZBasic.
KEYBOARD	DTEST.BAS	Configuration for keys.
GRAPHICS	PYRAMID.BAS	3-D pyramid rotates in space using standard graphics.
	HOUSE.BAS	3-D house rotates in space.
	SIEVE.BAS	The Sieve of Erastothenes benchmark from Byte, Jan. 1983.
	COLOR.BAS	Shows different uses of color.
	PAGEFLIP.BAS	Demonstration of graphics page flipping in EGA modes.
	ZROSE.BAS	Does a graphic "Rose" using High-speed and regular speed SIN and COSINE routines. Change mode for different graphics types (MODE 7 for CGA, MODE 19 for EGA, MODE 20 for Hercules and Hercules Plus).
	GEDIT.BAS	A powerful graphic editor. Requires a Microsoft mouse.
	MODE.BAS	This program displays some of the MODE type available in ZBasic. It does not demonstrate EGA or Hercules modes. Extend the loop from 16 to 20 to do that.
	APPLE.BAS	Bar and Line Graphs in Device independent Graphics.
GAME	KILLER.BAS	A simple game written in ZBasic. It is quite fun. Try it.
SORTS	SORT.BAS	This routine creates random data for arrays to demonstrate the SHELL and QUICK sort routines on this disk. Load this program first then do `APPEND 1000 SHELL.APP (or QUICK.APP)`
	SHELL.APP	The SHELL SORT that appears in the manual (under "Array variables.) A powerful sort when less items are used.
	QUICK.APP	The QUICK SORT that appears in the manual (under "Array variables.) A powerful sort when many elements need to be sorted.
MATH FUNCTIONS	SCIFN.APP	Examples of creating your own functions with ZBasic.

MSDOS APPENDIX

SPECIAL NOTES

LOADING OLD ZBASIC 3.02 PROGRAMS

To convert old ZBasic 3.02 programs to work with ZBasic version 4.0:

1. Load programs into ZBasic 3.02
2. Use SAVE* to save them in ASCII format
3. Load them into ZBasic 4.0

Note: Be sure that "Spaces between Keywords" is set to "NO" when loading old programs (otherwise keywords without spaces will result in syntax errors [or worse]).

PATHNAMES

The filenames in ZBasic are the standard MSDOS filenames as specified in the MSDOS reference manual. Pathnames may not be used as filenames within OPEN statements in this version (although by using CHDIR you may have files open in many different directories simultaneously).

To control pathnames use one of the following: PATH$, RMDIR, MKDIR and CHDIR and ERROR 11. See the listings for these commands in this appendix.

NOT QUITE 100% IBM PC COMPATIBLE NOTES

For many computers that are not quite 100% compatible with the IBM PC there are a number of things you may want to do. First; be sure to read the text in the "MSDOS Specific Configuration Options" section of this manual. Especially under IBM Text Compatible and IBM Graphics Compatible items. You may need to set these. Other than that you should have few problems using ZBasic on your computer.

NOTE TO TANDY 2000 OWNERS

The Tandy 2000 boots up with random characters on the screen. While this may look like the system has crashed, it is alive and well on another page. Simply press "E" and everything works fine from there on out. Use CONFIG from the Standard Line Editor to configure ZBasic to your preferences.

ZBasic operates normally on all the other Tandy computers including the Tandy 1000, 1200 and 3000.

NOTE TO ZENITH Z-150 OWNERS

Zenith Z-150 computers work great with ZBasic when you use the PC emulation program available from Zenith and Heath dealers nationally. Failure to use this program produces a "Wild Interrupt Error".

MSDOS APPENDIX

ADDITIONS TO THE STANDARD LINE EDITOR

OPTIONAL KEYS

The standard reference manual describes certain keys to be used with the Standard Line Editor. The following list of keys may also be used:

KEY		DEFINITION
Up arrow	--->	List previous line
Down arrow	--->	List next line
Home	--->	List first line
End	--->	List last line
Page down	--->	List next 10 lines
Ctrl-Home	--->	Clear the screen
Cursor keys	--->	Cursor movement left or right (or <SPACE> and <BKSPACE>)
Insert	--->	Enter Insert Mode (or I)
Delete	--->	Delete Characters (or D)

Also see "Full Screen Editor" in the back of this appendix.

ADDITIONAL COMMANDS

Several new commands and some helpful screen displays have been added to make the Standard Line Editor easier to use and more powerful.

KEY ON
KEY OFF
KEY LIST

The commands; KEY ON, KEY OFF and KEY LIST have been added to give you control of the function key command options that you've noticed are now displayed on the bottom of the screen. KEY OFF will "hide" the display, KEY ON will "show" the display and KEY LIST will list the commands associated with the function keys.

See KEY in this appendix for details.

FIX [number of spaces to indent]
INDENT [number of spaces to indent]
RENUM*
UNNUM

These commands are added to allow more control over source code displays.

For those folks that don't like line numbers, the UNNUM command has been added. It strips the program of line numbers. RENUM* adds line numbers back. INDENT forces the text to be indented when you go into the Full Screen Editor. FIX does both UNNUM and INDENT at the same time.

See FIX, INDENT, UNNUM and RENUM* in this appendix for details.

MSDOS APPENDIX

GRAPHIC ENHANCEMENTS

This new version of ZBasic supports the following graphics adaptors:

EGA: Enhanced Graphic Adaptor	MODE 16, 17, 18, 19 (and CGA)
CGA: Color Graphic Adaptor	MODE 0-15
MDPA: Monochrome Display and Printer adaptor	MODE 2
HERCULES Monochrome Graphic Board	MODE 20
HERCULES Plus Monochrome Graphic board	MODE 20

Note: See CARDTYPE statement in this appendix for determining board installed.

NEW GRAPHIC COMMANDS

In order to use graphics with the appropriate boards you will need to reference the following statements and functions in this appendix and in the main reference section:

This Appendix		Main Reference
SCREEN	TFORMAT	COORDINATE
PALETTE	CARDTYPE	COORDINATE WINDOW
CSRLIN	PLOT USING	GET
MODE (15-20)	PAINT	PUT

HERCULES™ and HERCULES PLUS™

This new version of ZBasic supports the Hercules and Hercules Plus Monochrome graphics boards when in MODE 20.

In order to use text AND graphics in MODE 20 you must load ZBasic by typing ZHERC from the MSDOS command line. This loads the Hercules text driver below ZBasic and will not interfere with other drivers.

You must also include this driver with stand-alone programs you create with ZBasic if they require text and graphics in Hercules MODE 20. Simply create a BATCH file with the following code to do this. Change *filename* to the name of your program:

```
HERC.COM
filename.COM
```

This loads the Hercules character driver into memory and allows ZBasic to access it properly.

Hercules graphics are supported without any additions. That is; if you only want to use the 720x348 Hercules graphics without text, you may use graphics in MODE 20 without loading the HERC.COM driver into memory. You may use text in the other modes with the highest quality text in MODE 2.

TFORMAT is used to control the format for placing text on the Hercules graphics screen. Options include Reverse, regular and XOR modes. See TFORMAT in this appendix for specifics.

MSDOS APPENDIX

EXECUTING ZBASIC FROM MSDOS

There are basically three ways of starting ZBasic from the operating system:

1. **Type:** `ZBASIC <ENTER>`

This is the standard way to startup ZBasic. See "Getting Started" in the front of this manual and also "MSDOS Specific Configuration Options" in this appendix.

2. **Type:** `ZBASIC filename <ENTER>`

This will force ZBasic to load the file given by *filename* and go directly into the ZBasic editor, skipping the initial prompt screen. Using this procedure saves the time of going through the initial prompt screen (note: BASICA or QuickBASIC files must have been saved in ASCII format).

3. **Type:** `ZBASIC filename1 filename2 [/C] <ENTER>`

This will force ZBasic to load the file given by *filename1*, then compile it into the file given by *filename2*. If the "/C" option is included, then the file will be saved as a ZBasic chain file (same as using RUN+), else the file will be saved as an executable file (same as using RUN*).

ZBasic will always return to the operating system when done compiling, making this procedure very useful for compiling several programs at once using a batch file.

NOTE To convert old ZBasic 3.02 programs saved in tokenized format so they can be loaded into the new version 4.0:

1. Load programs into ZBasic 3.02
2. Use SAVE* to save them in ASCII format
3. Load them into ZBasic 4.0

EXAMPLES

`ZBASIC TEST.BAS TEST.COM`
This compiles `TEST.BAS` and creates `TEST.COM`; an executable file.

`ZBASIC TEST.BAS TEST.CHN/C`
This compiles `TEST.BAS` and creates `TEST.CHN`; a chain file (runtime not included).

If there is an error during loading or compiling, the error will be displayed and ZBasic will return to the operating system. ZBasic will also return an exit code of 1 if there was an error, 0 if your program compiled and saved successfully. This exit code can be examined using the batch sub-commands IF and ERRORLEVEL.

See the MSDOS manual for more information on using batch files.

MSDOS APPENDIX

MSDOS SPECIFIC CONFIGURATION OPTIONS

ZBasic may be configured by typing "C" in the initial prompt screen. ZBasic will then ask for the standard configuration parameters explained in the "Configure" section of the manual (this version may also be configured directly from the editor by typing CONFIG).

Following the "Standard configuration parameters" are the MSDOS specific configuration parameters. The additional configuration prompts are:

Default BASIC file type BAS ?

Allows you to set the default suffix. If you save a program called FRED, it will be saved as FRED.BAS. If you change BAS to ZBS, it will be saved as FRED.ZBS. Use a space if no suffix is desired. The default is .BAS.

Default screen length 0019 00025 ?

The normal IBM screen allows 25 rows of text to be displayed. There are some MSDOS computers, however, that cannot display this many rows of text. For these computers, simply enter the actual screen length under this prompt and ZBasic will automatically correct this IBM incompatibility.

DELAY 1ms time constant 012C 00300 ?

The ZBasic DELAY *expression* statement is designed to delay a number of milliseconds given by *expression.* However, each millisecond delay is dependent on the speed of the computer hardware.

ZBasic assumes the computer speed to be that of the IBM PC (i.e. 4.77 megahertz clock speed using the 8088 microprocessor). If your clock-speed varies; enter a number under this prompt corresponding to your computer's speed: For 8mhz calculate using `8/4.77*300=503`. For 12mhz calculate using `12/4.77*300=754`

This 1ms time constant is also used in the ZBasic SOUND statement; thus, if a program uses sound at all, it is necessary that this time constant be accurate. The time constant can also be changed during program execution using the pre-defined user function -- USR2. See USR2 statement in this appendix for more information.

Mouse supported <Y/N>. N ?

ZBasic defaults to not supporting a mouse driver. If ZBasic is configured to support the mouse, it will assume the Microsoft mouse and will always check to see if the mouse hardware and software are installed.

On some machines, this check is invalid and can cause unpredictable results. This is because ZBasic checks and, if non-zero, uses interrupt H33 for the mouse interface. If a system uses this interrupt vector for something else, problems will result.

If configured to use the mouse, make sure to test ZBasic with the mouse driver.

MSDOS APPENDIX

IBM graphics compatible <Y/N>. Y?

Graphics in MODE 5 and MODE 7 go directly to memory and, thus, are very fast. This is a problem, however, on systems with the graphics memory arranged differently from the IBM PC's graphics. If this is the case, enter "N" under this prompt and ZBasic will no longer use direct memory when implementing graphics (which will noticeable slow down the graphics functions).

Selecting the "N" option will also allow ZBasic to handle COLOR ranges from 0 to 255 for foreground, background, and palette in MODE 5 and MODE 7 (see "COLOR" in this appendix). This will allow the full color range on some advanced color adapters; however, the expressions used in the COLOR statement should not exceed the highest allowed value for the adapter, else the result will be unpredictable.

IBM text compatible <Y/N>. N?

If ZBasic is configured to be IBM text compatible, then ZBasic writes text directly to memory. This speeds up the PRINT statement by as much as ten times, depending on what is being printed. If the machine is not a true compatible, however, ZBasic will not operate properly.

To find out if this works on your machine, simply type **"Y"** after the question mark; if the next configuration parameter shows up normally on the screen, then the machine is IBM text compatible. A program that is compiled to disk using 'RUN*', however, cannot be configured to be IBM text compatible. This is done as a safety measure to insure that a compiled ZBasic program will run on different machines. A program can set itself to this configuration by using:

POKE &342,1 <--- sets to IBM text compatible (for high speed text printing)
POKE &342,0 <--- sets to non-IBM text compatible (prints through BIOS)

A program should allow the user to configure the program to his machine.

LOCATE order is X,Y <Y,N>. Y?

ZBasic assumes an orientation of X,Y in the LOCATE statement, which corresponds to column,row. This is different from MSBASIC. Thus, if converting a program from a BASIC using Y,X orientation, enter "N" under this prompt and ZBasic will then use this representation. This will not, however, change the orientation on the ZBasic PRINT@(X,Y) statement.

LOCATE start is 0,0 <Y/N>. Y?

The LOCATE statement in IBM BASIC uses 1,1 as the upper left hand corner of the screen, but ZBasic uses 0,0. If "N" is entered under this prompt, ZBasic will use the same convention as IBM BASIC. Notice that both the "LOCATE start" and "LOCATE order" configuration parameters must be changed to "N" in order for the LOCATE statement to operate as IBM BASIC.

MSDOS APPENDIX

MEMORY USE WITH VERSION 4.0

The MSDOS version of ZBasic is designed to run on the IBM PC and most compatibles under MSDOS 2.0 or greater.

For those Not-So-Compatible-Compatibles, see "MSDOS Specific Configuration Options" in this appendix for ways of configuring ZBasic to work with your computer.

At least 128 k of memory is required for editing and compiling of programs, although ZBasic compiled programs can be written to run on 64K systems. See "Memory Considerations" in this appendix for more information.

MEMORY FOR VARIABLES

SINGLE AND DOUBLE PRECISION

This version of ZBasic offers extended capacities for Single and Double precision arrays. Arrays are limited to available memory up to 640K. The array may be larger than 64K. See MEM BCD function for determining where the regular BCD variable segment begins (not arrays). See FRE to determine available memory.

Regular Single and Double precision variables (not arrays) and regular string variables are limited to 64K total.

STRING

String arrays may be larger than 64K in this version. See MEM STR function for determining where segment begins for string arrays.

Regular Single and Double precision variables (not arrays) and regular string variables are limited to a total of 64K.

INTEGER

Integer variables may use a maximum of 64K memory. This includes all integer arrays and regular integer variables. If you need an integer array larger than 64K you can use either a floating point array or use a string array and store and retrieve the integer numbers in the string array with CVI and MKI$. This will take only 2 bytes per element.

OTHER MEMORY CONSIDERATIONS

The MSDOS version of ZBasic has three different modes of operation concerning memory organization -- EDIT mode, RUN mode, and RUN* mode (see memory map on following page). At least 128k of memory is required for the EDIT and RUN modes (the development stage of the program). However, after a program has been compiled and saved using RUN*, it can be run on as little as 64k of memory depending on the size of the program (the RUN* mode is shown on the memory map).

continued...

continued from previous page

While in EDIT mode, ZBasic will assume to own all of the existing memory available in the machine. Thus, if another program (i.e. a .COM or .EXE file) attempts to use parts of memory located above (higher address than) ZBasic, the contents of this memory could be destroyed by ZBasic. A "Memory Allocation Error" may also be generated by the operating system in this case (because a memory block created by the other program will have been written over). Thus, if any drivers are to be resident in memory, they should be installed before ZBasic is given control.

A program compiled by ZBasic as a .COM file only uses the amount of memory required by the program. This means that other programs can use the memory outside of the ZBasic compiled program. If a ZBasic program chains to another program, ZBasic will try to re-size its current memory block to fit the chained program. If the new program is larger and does not fit, ZBasic will not execute the chain and will return a disk error.

Note: See VARPTR for details about locating variables in memory.

NOTES ON ZBasic MEMORY MAP:

1. The MEM type functions are used to return the address of a segment break.

MEM C	CODE segment
MEM D	Disk buffer and INTEGER variable segment. Integer variables (including arrays) and disk buffers may use up to 64K total.
MEM E	EXTRA segment
MEM S	STACK segment
MEM I	INDEX$ segment
MEM BCD	The segment address for the BCD "simple" variables (non-array). Up to 64K total memory may be used (single or double precision variables).
MEM STR	The Segment address for the String "simple" variables (non-array). Up to 64K total memory may be used.
MEM ARR *var(n)*	The segment address for the String or BCD array specified by *var* (n is a dummy expression or number and is not used).

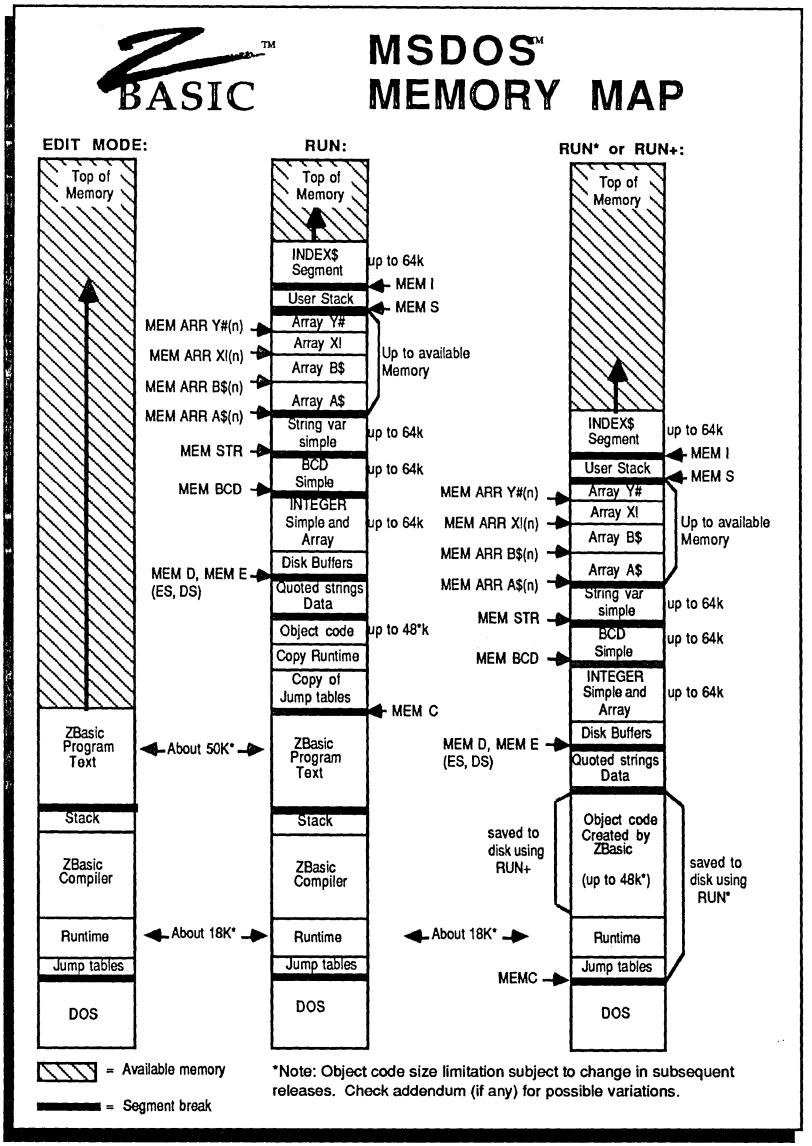
For more information, see "MEM function" in this appendix.

2. The ZBasic subroutines and jump tables are not saved to disk when a program is compiled as a chain file using RUN+. Thus, chain files take about 18-19k less on disk.

3. ZBasic is not necessarily located immediately after MSDOS. There may be drivers or other applications installed before ZBasic. ZBasic does, however, assume to own all of the memory above it.

4. The size of the INDEX$ segment is determined by the CLEAR statement (see reference section in main manual). The MSDOS version of ZBasic defaults to CLEAR 1024, making the INDEX$ segment 1k. If there is not enough memory to create a segment of the size specified, then the largest size available will be allocated. The size of the INDEX$ memory can be determined using the MEM function.

5. When the CALL *string* or SHELL *string* statements are used to load and execute another program , the program is loaded just above the ZBasic INDEX$ segment (see CALL and SHELL in this appendix .


ZBASIC™
MSDOS™
MEMORY MAP
EDIT MODE:
RUN:
RUN* or RUN+:
Top of Memory
INDEX$ Segment
User Stack
Array Y#
Array XI
Array B$
Array A$
String var simple
BCD Simple
INTEGER Simple and Array
Disk Buffers
Quoted strings Data
Object code
Copy Runtime
Copy of Jump tables
ZBasic Program Text
Stack
ZBasic Compiler
Runtime
Jump tables
DOS
up to 64k
MEM I
MEM S
Up to available Memory
MEM ARR Y#(n)
MEM ARR XI(n)
MEM ARR B$(n)
MEM ARR A$(n)
MEM STR
MEM BCD
MEM D, MEM E (ES, DS)
up to 48*k
MEM C
About 50K*
About 18K*
Object code Created by ZBasic
(up to 48k*)
saved to disk using RUN+
saved to disk using RUN*
MEMC
= Available memory
= Segment break
*Note: Object code size limitation subject to change in subsequent releases. Check addendum (if any) for possible variations.

MSDOS APPENDIX

RS-232 COMMUNICATION

ZBasic for MSDOS supports asynchronous communication using the numbers; -1 for COM1 and -2 for COM2 within the OPEN"C" statement. Baud rate, parity, stop bits, word length, handshaking and buffer length are all controlled in the OPEN "C" statement.

The OPEN "C" statement has additional parameters that can be used to control the handshaking on the RS-232 cable when writing to the port. See OPEN "C" in this appendix and the example routines under OPEN"C" in the main reference section.

To switch the communication interrupts on or off, the commands COM ON and COM OFF may now be used. The buffer size may be up to 32,700 bytes. See COM ON and COM OFF in this appendix for details. To determine the status of the buffer see COM BUFF in this appendix.

To trap communication errors see ON COM ERROR GOSUB in this appendix.

There is also a predefined user function -- USR5 -- that returns the modem and line status for the asynchronous adapter; see USR5 function in this appendix.

COMMON COMMUNICATION PROBLEMS

If the asynchronous communication is not working properly, try any of the following:

1. Check to make sure the baud rate, parity, stop bits, and word length settings are the same on both sides of the communication. See ON COM ERROR GOSUB in this appendix.

2. Examine the modem and line status. Type in the program below to observe the status of the asynchronous port:

```
CLS
DO
  PRINT@(0,0) BIN$(USR5(-1));: REM COM 2 is -2
UNTIL LEN(INKEY$):  REM  Press a key to stop
```

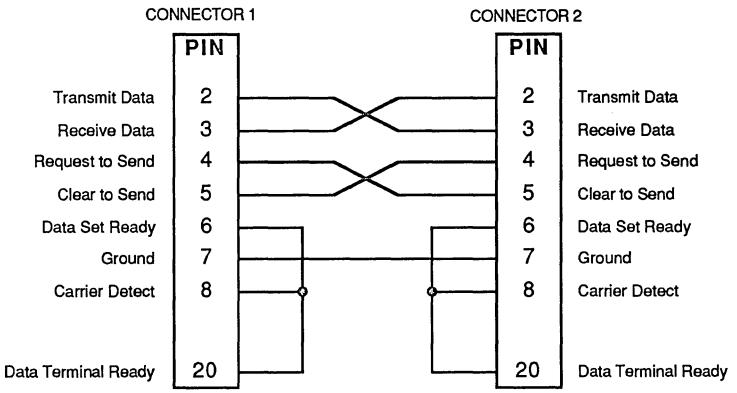
   If using COM2, then use USR5(-2). The meaning of the bits are explained in this appendix under USR5 (bit 15 is on the left, bit 0 on the right). A Framing error or Parity error usually means that the sender and receiver are operating at different baud rates, parity, stop bits, or word length. An overrun error usually indicates an improper cable or buffer length has been set to small.

3. Check for proper cable. The cable must support the standard RS-232 asynchronous interface. If the serial transfer works at a low baud rate (like 300 baud), but fails at higher baud rates, then the cable is improper. The diagrams on the next page show the two most typical cable configurations. The top diagram is for communication between two DTE's (Data Terminal Equipment) or two DCE's (Data Communication Equipment). This configuration is typical for an IBM talking to another IBM or compatible. The bottom diagram is for communication between a DTE and a DCE. These cable configurations are not the "rule", they are only the most typical for proper RS-232 interface.

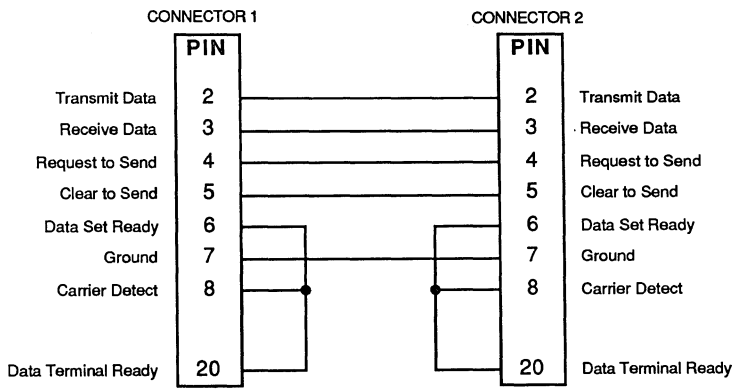
continued...

RS-232 COMMUNICATION continued

Communication devices: DTE <----> DTE
DCE <----> DCE
(Typical for IBM to IBM communication)



Communication devices: DTE <----> DCE
DCE <----> DTE



DTE: Data Terminal Equipment
DCE: Data Communication Equipment

MSDOS APPENDIX

ZBasic JUMP TABLE

The MSDOS version of ZBasic makes a jump table available starting at address 0103 hex. These jumps can be altered to jump to some other routine to handle the same function. This can be useful for implementing special hardware or for handling non-compatible DOS or BIOS.

Also included in this section are the USR function jumps, many of which are predefined. The following is a list of all the available jump locations with a short description of each:

LIST OF ALTERABLE JUMPS:

Address	Description
&103	ZBasic exit - where ZBasic jumps on a STOP or END statement. On entry: ---- On exit: exit program...
&106	Video output - all characters output to the screen. On entry: AL = character On exit: ----
&109	Printer output - all characters output to the printer. On entry: AL = character On exit: ---- Remarks: altering will disable ZBasic PAGE control
&10C	Scan keyboard - used by INKEY$ and TRON commands. On entry: ---- On exit: Z flag if no character NZ flag if character in AL Remarks: SI must be preserved
&10F	Init. COM port - called by the OPEN "C" statement. On entry: ---- On exit: ---- Remarks: sets Baud,Parity,Stopbits, and Wordlength
&112	Write COM port - used whenever filenumber is #-1 or #-2. On entry: AL = character On exit: ---- Remarks: CX,DX must be preserved
&115	Read COM port - used whenever filenumber is #-1 or #-2. On entry: ---- On exit: AX = character Remarks: CX,DX must be preserved

LIST OF ALTERABLE JUMPS continued

Address	Description
&118	Scan COM port - used by READ #-1,A$;0 (same as INKEY$, but for COM). On entry: DS:SI points to destination string On exit: DS:SI contain string in form *length,characters*
&11B	SOUND *frequency,duration* - ZBasic statement. On entry: AX = duration BX = frequency On exit: ...sound...
&11E	MOUSE (*expression*) - ZBasic function (not statement). On entry: AX = expression On exit: AX = value returned
&121	Write dot - sets each graphic point in MODE 5 and MODE 7 when ZBasic is configured to not be IBM graphics compatible (see "MSDOS CONFIGURATION" in this appendix). On entry: AX = row number BX = column number On exit: ---- Remarks: the row and column are actual screen pixels
&124	POINT function - reads the color of a screen point. On entry: AX = row number BX = column number On exit: AX = color of point Remarks: the row and column are actual screen pixels
&127	Convert screen coordinates - converts ZBasic X,Y to pixel col,row. On entry: AX = ZBasic Y (or row) BX = ZBasic X (or column) On exit: AX = screen row number BX = screen column number
&12A	PLOT USING statement. On entry: AX = row number BX = column number CL = magnitude SI points to string On exit: ---- Remarks: the row and column are actual screen pixels

MSDOS APPENDIX

LIST OF ALTERABLE USR FUNCTION JUMPS

User functions using USRn(*expression*) statement.
On entry: AX = *expression*

Address	Description
&12D	USR0 - not predefined
&133	USR1 - predefined ---> End Of File function (same as EOF)
&139	USR2 - predefined ---> DELAY time constant
&13F	USR3 - predefined ---> keyboard functions
&145	USR4 - predefined ---> set break vector
&14B	USR5 - predefined ---> read COM port status
&151	USR6 - predefined ---> last line number recorded
&157	USR7 - predefined ---> random number generator
&15D	USR8 - predefined ---> integer sine
&163	USR9 - predefined ---> integer cosine

NOTES ON USING THE ZBasic JUMP TABLES:

1. All routines must exit with the segment registers DS, ES, and SS the same as that upon entry.

2. Exiting the routine is done with a RET assembly opcode (or a RETURN ZBasic statement).

3. The jumps are all relative to the code segment on the 8088/86 processor, so the actual value to poke into the jump table has to be calculated. The program below is an example of how to change a jump vector.

```
REM EXAMPLE TO CHANGE SOUND TO LINE 1000
Jump = &11B
Line = LINE 1000
POKE WORD Jump+1, Line-(Jump+3), MEMC
```

   This program can easily be modified to change any of the jump vectors to any ZBasic line number.

4. The MSDOS version of ZBasic contains a 30 byte patch area that can be used to contain a small routine. By using the patch utility (from the ZBasic startup screen), a jump vector can be permanently changed to jump directly into a routine written into the patch area.

Patch area address:	start	--->	&169	
	stop	--->	&187	(30 bytes)
Additional patch area:	stop	--->	&1B9	(49 bytes more)

   This additional patch area (49 bytes) can only be used if the ON INKEY$ statement is not implemented. (For more information on ON INKEY$, see "ON INKEY$" in this appendix.)

MACHINE LANGUAGE EXAMPLES

Following is a simple machine language example that instructs the operating system to print a ZBasic string variable. The operating system is accessed through a DOS function call (see DOS manual for more information on DOS function calls).

The assembly listing below is implemented using the MACHLG statement given in the ZBasic listing. Notice how the address of the variable A$ is generated in the MACHLG statement -- ZBasic automatically inserts the address of the variable when it is specified.

---------- **ASSEMBLY LISTING** ----------

```
0000                CSEG        SEGMENT PUBLIC 'CODE'
                    ASSUME      CS:CSEG, DS:DSEG

                    ;MAKE MSDOS FUNCTION CALL TO PRINT STRING
                    ;(must be terminated with '$')

0000                            ORG   0
0000 BA 0000        PRINTS:     MOV   DX,OFFSET String
                                ;Get string address in DX

0003 42                         INC DX      ;Skip length byte
0004 B4 09                      MOV  AH,9
0006 CD 21                      INT  21H    ;Make DOS call 9

0000                DSEG                    SEGMENT PUBLIC 'DATA'

0000 01             String      DB  256  DUP(?)

0008                DSEG        ENDS
0008                CSEG        ENDS
                                END
```

---------- **ZBasic LISTING** ----------

```
CLS
A$ = "PRINT ME"
A$ = A$ + "$"  :REM  Must be terminated with "$"
REM ----- MAKE CALL TO DOS TO PRINT A$ -----
MACHLG &BA,A$,&42,&B4,&09,&CD,&21
```

The next two pages give a more involved example of the MACHLG statement. The example modifies interrupt 24 hex to jump to a ZBasic subroutine. This interrupt is used for control whenever a critical error occurs within DOS (such as the disk drive door being open during a read).

MSDOS APPENDIX

Assembly Listing of Routines to Control the Critical Error Handler:

```
0000                    CSEG        SEGMENT PUBLIC 'CODE'
                                    ASSUME CS:CSEG, DS:DSEG

                        ;           GET CRITICAL ERROR HANDLER VECTOR
                        ;           ON EXIT _Seg:_Offset= Address of error handler

0000                                ORG             0
0000 06                 GETVEC:     PUSH            ES      ;Save ZBasic's ES
0001 B8 3524                        MOV             AX,3524H
0004 CD 21                          INT             21H             ;DOS function call 35H - GET VEC
0006 8C 06 0006 R                   MOV             _Seg,ES         ;Save current vector
000A 89 1E 0004 R                   MOV             _Offset,BX
000E 07                             POP             ES

                        ;           SET CRITICAL ERROR HANDLER VECTOR
                        ;           ON ENTRY:       _Seg = ZBasic's code seg
                        ;                           _Offset = LINE # of error handler
0000                                ORG             0
0000 1E                 SETVEC:     PUSH            DS      ;Save ZBasic's DS
0001 8B 16 0004 R                   MOV             DX,_Offset      ;Get vector in DS:DX
0005 8E 1E 0006 R                   MOV             DS,_Seg
0009 B8 2524                        MOV             AX,2524H
000C CD 21                          INT             21H             ;DOS function call 25H - SET VECT
000E 1F                             POP             DS

                        ;           NEW CRITICAL ERROR HANDLER VECTOR
0000                                ORG             0
0000 53                 ERRVEC: PUSH                BX              ;SS,SP,DS,ES,BX,CX,DX must
0001 51                             PUSH            CX              ;be preserved!
0002 52                             PUSH            DX
0003 1E                             PUSH            DS
0004 06                             PUSH            ES
0005 8B EC                          MOV             BP,SP
0007 8E 5E 1E                       MOV             DS,[BP+30]      ;Get ZBasic's DS and ES
000A 8E 46 20                       MOV             ES,[BP+32]
000D 89 3E 0000 R                   MOV             Error,DI        ;Put error code into variable

                        ;         END OF NEW CRITICAL ERROR HANDLER VECTOR
0000                                ORG             0
0000 A1 0002 R                      MOV             AX,Response     ;Put decision in AL
0003 07                             POP             ES              ;Restore registers
0004 1F                             POP             DS
0005 5A                             POP             DX
0006 59                             POP             CX
0007 5B                             POP             BX
0008 CF                             IRET                            ;Return to DOS...

0000                    DSEG        SEGMENT PUBLIC 'DATA'

0000 01                 Error       DW              1  DUP(?)
0002 01                 Response    DW              1  DUP(?)
0004 01                 _Offset DW                  1  DUP(?)
0006 01                 _Seg        DW              1  DUP(?)
0008                    DSEG        ENDS
0008 CSEG                           ENDS
                                    END
```

ZBasic Listing to Control Critical Error Handler:

```
REM ----- GET PREVIOUS CRITICAL ERROR VECTOR -----
MACHLG &06,&B8,&3524,&CD,&21,&8C,&06
MACHLG Old_Seg%,&89,&1E,Old_offset%,&07
:
REM ----- SET CRITICAL ERROR VECTOR TO LINE 1000 -----
Seg%=MEMC : Offset%=LINE "Error routine"
MACHLG &1E,&8B,&16,Offset%,&8E,&1E,Seg%,&B8,&2524,&CD,&21,&1F
:
REM ----- READ DRIVE A: WITH DOOR OPEN -----
OPEN "R",1,"A:HELPME"
:
REM ----- RESET ERROR VECTOR BEFORE EXIT TO ZBASIC -----
MACHLG &1E,&8B,&16,Old_offset%,&8E,&1E
MACHLG Old_Seg%,&B8,&2524,&CD,&21,&1F
STOP
:
:
"Error  routine"
REM   START OF ERROR HANDLER ROUTINE
REM   ****************************
:
MACHLG &53,&51,&52,&1E,&06,&8B,&EC,&8E
MACHLG &5E,&1E,&8E,&46,&20,&89,&3E,Error%:
:
REM ----- DO ANYTHING HERE EXCEPT DISK I/O -----
:
PRINT
PRINT"Error. What do you want to do"
PRINT"(Ignore, Retry, Terminate)? ";
DO
  A$=INKEY$
UNTIL LEN(A$)
PRINT A$ : A$=UCASE$(A$)
I%=INSTR(1,"IRT",A$) : IF I%=0 THEN 1060
Response%=I%-1
:
"Do DOS"
REM ----- RETURN TO DOS -----
MACHLG &A1,Response%,&07,&1F,&5A,&59,&5B,&CF
REM ************************************************
```

NOTES ON EXAMPLE

1. The ZBasic code that handles the critical error (lines between "Error routine" and "Do DOS" above) cannot use any DOS function calls greater than 12H. ZBasic uses these function calls on the following:

   a. All DISK I/O!
   b. TIME$ function and statement
   c. DATE$ function and statement
   d. CALL *string* statement
   e. END = *expr* statement

2. See YOUR MSDOS manual for more information on the critical error handler vector.

MSDOS APPENDIX

CONVERTING OLD BASICA™ or QuickBASIC™ PROGRAMS TO COMPILE WITH ZBasic™

First; Read the chapter in the front of this manual called "Converting Old Programs".

Be sure to set the the following options under "Configure" before loading your old QuickBASIC, BASICA or other MSBASIC programs:

• Default variable type:	**S** (to increase speed: avoid doing this)
• Convert to Uppercase Y/N:	**Y**
• Optimize expressions for Integer Y/N:	**N**
• Spaces required after keywords Y/N:	**Y**
• LOCATE x,y Y/N:	**N**
• LOCATE start is 0,0 Y/N:	**N**

WHAT DO WE HAVE THAT THEY DON'T?

- Hercules and Hercules Plus Graphics support.
- String arrays may be greater than 64K; up to available memory.
- Floating point arrays may be larger than 64k; up to available memory.
- No String "Garbage Collection". Static and dynamic String allocation.
- INDEX$ string array and special commands: INDEX$I, INDEX$D and INDEXF.
- Direct commands from the Standard Line Editor.
- Configurable BCD math up to 54 digits of precision.
- Device independent Graphics. See COORDINATE.
- Apple //, Macintosh and Z80 versions.

STRING LENGTH NOTE

QuickBASIC allows up to 32k for a string length. The maximum string length in ZBasic is 255. See notes under "Strings" in "Converting Old Programs" in the main reference section of this manual.

COMMANDS THAT ARE DIFFERENT

The following list of QuickBASIC commands are not completely compatible with ZBasic. Examples of possible alternatives are given.

CALL	Different syntax with ZBasic. See CALL in this appendix.
CDBL	Not applicable.
CHAIN	See CHAIN in the main reference section.
CIRCLE	Some variation in extra parameters.
CLEAR	Some syntax the same. Does not allow changing stack size.
COLOR	Some syntax differences. See COLOR.
COM STOP	COM STOP not supported. See COM ON /OFF.
COMMON	Same as ZBasic DIM . The SHARED parameter is not supported.
CSNG	Automatic when the target variable is a single precision number.
CVD, CVS	ZBasic uses CVB with both single and double precision BCD variables. It is not necessary to use this with ZBasic file commands since numeric variables may be used with READ# and WRITE# (see FIELD below).
DATA	Same except that strings beginning with numbers must be in quotes.

continued...

continued from previous page

DRAW	Not supported. See PLOT USING.
ENVIRON[$]	Not supported. See PATH$ and SHELL for alternatives.
ERASE	Not supported with ZBasic since all arrays are STATIC.
ERDEV[$]	See "Critical Error Handling" in this appendix.
ERR/ERL	Not supported.
ERROR	Not the same. See ERROR in this appendix and reference section.
EXIT	See END FN, END IF, END SELECT and END.
FIELD	Not supported. Fielding is done automatically in the READ# and WRITE# statements. There is no need to use MKI$, MKD$, MKS$, CVI, CVS, CVD, LSET or RSET to store or remove data from a field. Saves lots of time. See "Files" in the main reference section of this manual.
GET (files)	Not supported. See RECORD#
INPUT$	Not supported. See READ# *filenumber*, A$;*length*
IOCTL[$]	See USR5 in this appendix.
KEY n	See INKEY$ (n) statement in this appendix (ignored at runtime).
KEY LIST/ON/OFF	See KEY ON/OFF/LIST and ON INKEY$ in this appendix for variations. KEY ON and OFF in program code is ignored.
LBOUND	Since arrays are all static this is not necessary.
LINE	Use PLOT, PLOT TO, BOX, BOX FILL
LOCATE	X and Y are opposite, 0,0 is start. You may configure ZBasic for this.
LOCK/UNLOCK	Not supported.
LPOS	Use POS(1) instead.
LSET	See FIELD above.
MKD$, MKS$	See MKB$ in main reference section (BCD) and FIELD above.
ON COM	See COM ON/OFF, COM BUFF and ON COM ERROR GOSUB.
ON ERROR	Not the same. Disk errors only. See "Disk Error Trapping" in the front of the main reference section of this manual. Also see ON COM ERROR.
ON KEY$	See ON INKEY$ in this appendix.
ON PEN	See DEF MOUSE statement in this appendix.
ON PLAY	Not supported.
ON STRIG	See DEF MOUSE and MOUSE statement in this appendix.
ON TIMER	See TIMER in this appendix.
OPEN	The same in most respects. See OPEN in the main reference section. Pathnames are not allowed in filenames. See PATH$, CHDIR, RMDIR and MKDIR in this appendix.
OPEN COM	See OPEN"C" for syntax variations
OPTION BASE	The BASE of an array may be configured under "Configure".
PALETTE	Same except that PALETTE USING is not supported.
PCOPY	Not supported.
PEN	See DEF MOUSE and MOUSE function in this appendix.
PLAY	Not supported
PMAP	See DEF PAGE, VIEW PRINT and COORDINATE
PRESET	COLOR=background: PLOT x,y.
PSET	COLOR=foreground: PLOT x,y
PUT (files)	Not supported. See RECORD# and "Files" in reference section.
REDIM	Not supported since ZBasic uses static arrays.
RESTORE n	With ZBasic *n* restores to the *nth* item not the nth line.
RESUME	Not supported. See ON ERROR GOSUB in the main reference.
RND(x)	ZBasic returns an integer number from 1 to x. Not a number between zero and one. This method provides higher performance.
RUN filespec	See RUN, SHELL and CALL in this appendix.
SADD	See VARPTR and VARSEG in this appendix.
SCREEN	See SCREEN, MODE and COLOR in this appendix.
SHARED	See "Chaining" in the main reference.
SUB, SUB END	See LONG FN, END FN and APPEND.
STATIC	All arrays are already static so this is not needed.
STICK	See DEF MOUSE and MOUSE function to set for joysticks.
STRIG	See DEF MOUSE and MOUSE function to set for joysticks.
UBOUND	Not needed with ZBasic since all variables are STATIC.
WINDOW	See DEF PAGE and VIEW SCREEN.

MSDOS APPENDIX

MSDOS APPENDIX

ZBasic™ VERSION 4.0 MSDOS™
REFERENCE SECTION

The following pages contain commands, statements and functions included in the MSDOS version of ZBasic that are not necessarily included with other versions of ZBasic.

Also be sure to see the definitions of the following new commands in the main reference section:

BEEP
CASE
COMMON
COORDINATE
COORDINATE WINDOW
CSRLIN
END SELECT
EOF
GET and PUT (graphics)
NAME
PATH
RESET
SELECT

MSDOS APPENDIX

BLOAD statement

FORMAT **BLOAD** *filespec* [, [*offset*][,*segment*]]

DEFINITION Loads a block of memory that was saved as *filespec* using the BSAVE statement.

The optional *offset* is the position in memory to load the block and it will load into the current segment as defined using the DEF SEG statement (or the *segment* option if used).

If *offset* and/or the segment is omitted, the information that was stored in the file with BSAVE is used.

EXAMPLE

```
REM Loads the CGA screen saved with BSAVE on the next page.
:
MODE 7
:
BLOAD "CGASCRN.MEM"
DELAY 4000
```

REMARK See BSAVE and DEF SEG.

Address and lengths of different graphics memory:

MODE	TYPE	ADDRESS	LENGTH
0-15	CGA	&HB800	2048-16,384*
2	MDPA	&HB000	4,096
20	Hercules (page0)	&HB000	32,767
20	Hercules (page1)	&HB800	32,767

Note: EGA graphics are stored in memory in a different format than other graphic cards. To save images in EGA modes (16-19) you should GET the screen. Save the screen to disk using:

```
BSAVE filespec, VARPTR(var%(n)), number of elements/2
```

BLOAD the integer array back into memory using

```
BLOAD filespec, VARPTR(var%(n))
```

and then PUT the image back to the screen (see GET and PUT in the main reference section of the manual.

*See CGA, EGA , Hercules and MDPA technical manuals for more information.

BSAVE statement

FORMAT **BSAVE** *filespec* , *offset*, *length* [, *segment*]

DEFINITION Saves a block of memory as *filespec* so that it may be loaded later with BLOAD. It saves an exact image of memory.

The optional *offset* is the position in memory to load the block and will save the current segment as defined using the DEF SEG statement (or the *segment* option if used).

If DEF SEG is not used, the DATA SEGMENT (MEMD) is used.

EXAMPLE

```
REM This program saves a high-res CGA screen
REM which may be loaded with BLOAD on the previous page.
:
MODE 7: REM Adjust MODE for your computer.
:
PRINT "Hello there!!"
CIRCLE FILL 512, 383, 300
:
:
BSAVE "CGASCRN.MEM",0, 16384,&HB800
```

REMARK See BLOAD and DEF SEG.

Address and lengths of different graphics memory:

MODE	TYPE	ADDRESS	LENGTH
0-15	CGA	&HB800	2048-16,384*
2	MDPA	&HB000	4,096
20	Hercules (page0)	&HB000	32,767
20	Hercules (page1)	&HB800	32,767

Note: EGA graphics are stored in memory in a different format than other graphic cards. To save images in EGA modes (16-19) you should GET the screen. Save the screen to disk using:

```
BSAVE filespec, VARPTR(var%(n)), number of elements/2
```

BLOAD the integer array back into memory using

```
BLOAD filespec, VARPTR(var%(n))
```

and then PUT the image back to the screen (see GET and PUT in the main reference section of the manual.

*See CGA, EGA , Hercules and MDPA technical manuals for more information.

CALL statement

FORMAT **CALL** *address* [, *segment*]

DEFINITION This statement is used to execute a subroutine located in memory at the segment given by *segment* with an offset of *address* .

If *segment* is not given, then ZBasic's code segment is used.

EXAMPLE

```
CALL LINE "Routine"
```

Calls a ZBasic subroutine starting at "Routine" (same as GOSUB "Routine" but takes longer to execute).

```
CALL &H1ØØ,  &HØBD7
```

Calls a subroutine located at &HBD7 with an offset of &H100. This is a very dangerous use of the CALL statement.

REMARK Use caution when specifying the segment. Rarely is any subroutine loaded in the same place every time. The operating system will load a program into the lowest available address, which depends on other programs that may be resident in memory.

Any subroutine that is called by specifying the segment must return from the subroutine with a far return. Otherwise, unpredictable results will occur.

CALL statement (same as **SHELL**)

FORMAT **CALL** *string*

DEFINITION The CALL statement followed by *string* will load and execute another program or MSDOS command specified by *string*. If a null (empty) string is specified, then MSDOS will be loaded and executed, in which case typing EXIT in DOS will return to ZBasic.

This is identical to the "SHELL" statement and is retained to remain compatible with older versions of ZBasic. Also see SHELL in this appendix..

String must be either a string variable or a quoted string.

EXAMPLE

```
CALL  "DISKCOPY  A:  B:"
```
This will perform a diskcopy as if it was typed in from the DOS command line.

```
CALL  ""
```
Gives control to MSDOS; displays DOS prompt. Type `EXIT` to return to ZBasic.

```
CALL  "ZBASIC"
```
This will actually load and execute ZBasic. Typing QUIT will then return to the original ZBasic.

```
A$ = "DIR  A:*.BAS":  CALL  A$
```
This will get the directory of all .BAS files on the A drive.

REMARK This statement can also be very useful for executing batch files -- just use the name of the .BAT file (batch file) for *string.*

There must be at least 17k of memory free to use the CALL statement.

If the "COMMAND.COM" file is not found, the message "File not found" will be echoed to the display and control will be returned to ZBasic.

MSDOS APPENDIX

CARDTYPE function

FORMAT **CARDTYPE**

DEFINITION Returns the type of graphic hardware connected to the current system:

CARD TYPE	TYPE	CODE RETURNED
Color Graphics Card	CGA	0
Enhanced Graphics Card	EGA	1
EGA with Monochrome monitor	EGA-Mono	2
Hercules (or compatible)	HERC	3
Monochrome	MDPA	255

EXAMPLE

```
A= CARDTYPE
:
SELECT CASE A
  CASE 0
    PRINT"A CGA card is installed"
    MODE 7
  CASE 1
    PRINT"An EGA card is installed"
    MODE 19
  CASE 2
    PRINT"A monochrome monitor is connected to an EGA card"
    MODE 18
  CASE 3
    PRINT"A Hercules card is installed"
    MODE 20
  CASE 255
    PRINT"A Monochrome card (MDPA) is installed"
    MODE 2
  CASE ELSE
    PRINT"You can't read this!"
END SELECT

RUN

A Monochrome card is installed
```

REMARK This command can be useful in determining the display card of any system so that the MODE parameters in a program can be changed accordingly.

Combine this with the Device Independent Graphics that ZBasic offers and one can write a program that will work the same way on a variety of graphics adaptors.

Also see COORDINATE, COORDINATE WINDOW and the section in this appendix "Graphics Enhancements".

CHDIR statement

FORMAT **CHDIR** *pathname*

DEFINITION Changes the current directory to the directory specified by *pathname.* This enables ZBasic to access files in different directories.

A *pathname* is : drive: directory\ directory\...

EXAMPLE

```
A$ = PATH$(0):              REM gets current path
CHDIR "\ZBASIC\DATA":       REM changes path
OPEN "I" 1,"MYDATA.TXT ": REM opens file using new path
CHDIR A$:                   REM back to original path
```

REMARK Also see PATH$, RMDIR and MKDIR.

Note: IF an error is encountered when using CHDIR, RMDIR or MKDIR, ZBasic returns an error eleven (11) in the ERROR function. See ERROR, ON ERROR GOSUB and the Chapter "Disk Error Trapping" in the front of this manual.

CINT statement

FORMAT **CINT** (expression)

DEFINITION Same as INT. The difference between INT and CINT in BASICA and QuickBASIC is that CINT rounds up and INT rounds down.

Example:

CINT(99.5)=100 INT(99.5)=99

Use the function below to emulate the BASICA CINT statement.

EXAMPLE

```
REM This function emulates the BASICA CINT statement
:
DEF FN cint#( x#) = INT(x#+ (SGN (x#) * .5))
:
PRINT FN cint# ( 99.3), CINT(99.3)
PRINT FN cint# ( 99.5), CINT(99.5)
PRINT FN cint# (-99.5), CINT(-99.5)
END

RUN

 99             99
 100            99
-100           -99
```

REMARK See INT, FRAC and FIX in main reference section.

COLOR statement

FORMAT

Graphics MODES:
COLOR [=] [*foreground*] [, [*background*] [, *palette /blinking*]]

Character MODES:
COLOR [=] [*character*] [, [*attribute*] [, *border*]]

DEFINITION

This statement controls the color of all output to the screen. All of the parameters are optional. This statement acts quite differently between graphics and character MODE (EGA colors in MODE 18: 0=black, 1=low intensity, 2=blinking, 3=high-intensity).

Under all modes COLOR 0 will turn foreground off (black in B/W modes, space in character modes). COLOR -1 will set it to the brightest color (white in B/W modes):

CHARACTER MODES 0,2,4,6

character = 0-255	(represents an ASCII number)
attribute = 0-255	(see below and table 2 on the next page)
border = 0-15	(use table 1)

Examples of character formats available in most modes:

Normal Intensity	Statement	
Regular	COLOR ,7	
Blinking	COLOR,135	
Underline	COLOR,1	(Blue on some color monitors)
UL Blinking	COLOR,129	
Invisible	COLOR,136	(not all systems)

High Intensity	Statement	
Regular	COLOR,15	
Blinking	COLOR,143	
Underline	COLOR ,9	(Blue on some color monitors)
UL Blinking	COLOR,137	

Reverse (Inverse)	Statement
Regular	COLOR ,112
Blinking	COLOR,240

GRAPHIC MODES 1,3,5,7

foreground = 0-15	(use table 1 [limited by memory if using an EGA card])
background = 0-31	(not available in EGA modes)
blinking = 0 or not 0	

MODE 5 -

foreground = 0-3	(use table 3)
background = 0-15	(use table 1)
palette = 0,1	(use table 3)

MODE 7 -
foreground = 0 or not 0
background = not used
palette = not used

continued...

MSDOS APPENDIX

COLOR statement continued

Be sure to see the other sections in this appendix that cover graphics and color, including; COLOR, SCREEN, PLOT, CIRCLE, PALETTE, DEF PAGE READ/WRITE and MODE. MODE contains a chart of the resolutions, pages, memory and SCREEN numbers required for different boards.

TABLE 1: Border, background, and foreground colors

VALUE	COLOR	VALUE	COLOR
0	black	8	Gray
1	blue	9	Light Blue
2	green	10	Light Green
3	cyan	11	Light Cyan
4	red	12	Light Red
5	magenta	13	Light Magenta
6	brown	14	Yellow
7	white	15	White Intensified

Note: Text Borders are not used in EGA modes

TABLE 2: Attribute byte definitions

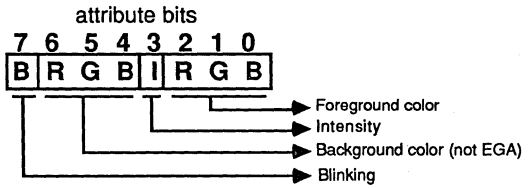


TABLE 3: Foreground colors

COLOR	PALLETE = 0	PALLETE = 1
0	background	
1	Green	Cyan
2	Red	Magenta
3	Brown	White

In Hercules mode there are only two colors; Black and White. There are three character formats: Regular, Reverse and XOR. See TFORMAT for more information.

COM BUFF function

FORMAT **COM BUFF** (*port*)

DEFINITION Returns the number of bytes storeing in the communication buffer.

Port	Device
-1	COM 1
-2	COM 2

The communications buffer is a first in, first out (FIFO) buffer . To read data out of the buffer use READ#, INPUT# or LINEINPUT#.

To set baud rate, buffer size, parity and handshaking, see OPEN"C".

EXAMPLE

```
OPEN "C", -1, 300,,,,,,32000:   REM COM (-1) ON is automatic
OPEN "O",1,"File.TXT"
:
program continues here...
LONG IF COM BUFF > 1000
  GOSUB"Read Buffer"
END IF
:
"Read Buffer"
DO
  READ#-1, A$;0                    <--- Reads even if buffer empty
  IF LEN(A$) THEN WRITE#1, A$;1    <--- Saves incoming data to a file
UNTIL COM BUFF=0
RETURN
END

RUN
```

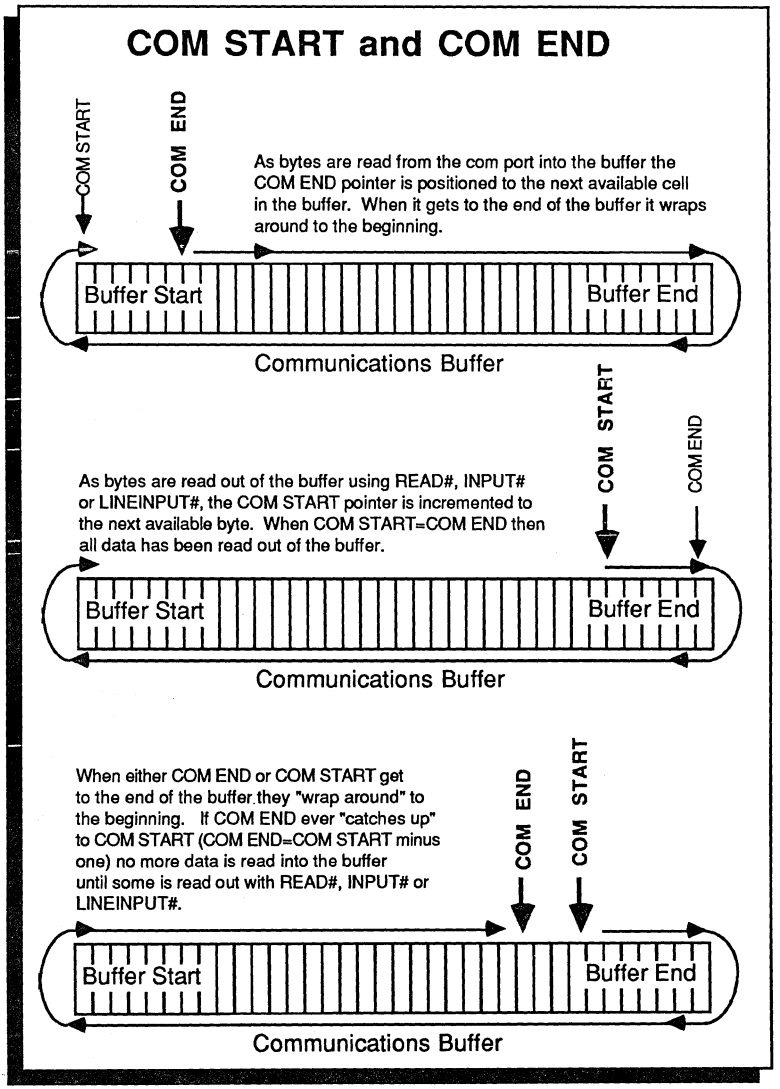
Prints incoming serial data to the capture file until the buffer is empty.

REMARK In this appendix see COM ON, COM OFF, OPEN"C", ROUTE and the chapter called "RS-232 communications". Also see in the main reference section ROUTE, OPEN"C", READ#, WRITE#, INPUT#, LINEINPUT# PRINT# and the chapter called "Files".

continued...

COM BUFF function continued

COM BUFF determines the size of the communication buffer in the following way; (COM END and COM START are imaginary pointers we use for illustration of how COM BUFF determines communication port size).



COM ON/OFF statements

FORMAT

COM (*port*) **ON**
COM (*port*) **OFF**

DEFINITION

Enables or disables the interrupt driven communications for a specified port. While program execution is taking place between these statements incoming data from the serial device specified by *port* is loaded into a FIFO buffer. It may be read out of the buffer using the READ#, INPUT# or LINEINPUT# statments.

Port	Device
-1	COM 1
-2	COM 2

COM ON is assumed when OPEN"C" is used.

EXAMPLE

```
OPEN "C", -1, 300,,,,,,32000: REM   COM (-1) ON is automatic.
program continues here...
:
:
DO
  READ#-1, A$;0
  PRINT A$
UNTIL COM BUFF=0
:
COM (-1) OFF
END
```

```
RUN
```

Prints incoming buffer data to the screen until the buffer is empty.

REMARK

See chapters in this appendix "RS-232 Communications", OPEN"C" and COM BUFF for other important information.

Note: The QuickBASIC or BASICA COM STOP statement is not supported. Use COM OFF instead.

COMMAND$ function

FORMAT **COMMAND$**

DEFINITION This function returns the command line string used to start up a program from DOS (ie. a ZBasic program saved as a .COM file).

The string does not include the name of the program.

EXAMPLE

```
"Myprog"
Message$=COMMAND$
PRINT Message$
END

RUN*                <--- Program saved as a stand-alone .COM file

Save as: Myprog.COM

A>Myprog  Hello there fred <--- Typed from MSDOS

Hello there fred
```

REMARK Also see "Executing ZBasic from MSDOS" in this appendix.

DATE$ statement

FORMAT **DATE$** = [*month*] [, [*day*] [, *year*]]

DEFINITION This statement is used to set the current date. Any of the three parameters can be omitted, in which case the parameter will not be changed. The following values are accepted:

month:	1 - 12
day:	1 - 31
year:	1980 - 2099

EXAMPLE

```
DATE$ = 8,20,1987
PRINT DATE$
:
DATE$=,1
PRINT DATE$

RUN

08/20/87
08/01/87
```

REMARK If any of the specified parameters are not in the accepted range given above, the date will not be changed. See TIME$ statement in this appendix and the TIME$ and DATE$ functions in the main reference section.

DEF MOUSE statement

FORMAT **DEF MOUSE** [=] *expression*

DEFINITION This statement sets the MOUSE function to return information from any one of four device drivers defined by *expression* :

DEF MOUSE	=	**0**	**MOUSE**
		1	**JOYSTICK A**
		2	**JOYSTICK B**
		3	**LIGHT PEN**

If the DEF MOUSE statement is not used, the mouse driver is used as the default.

If *expression* is not 0 to 3, then the MOUSE(*n*) function will always return zero.

EXAMPLE

```
DEF MOUSE=0
DO
  PRINT MOUSE(1), MOUSE(2)
UNTIL MOUSE(3)              <--- Press the Mouse button to stop.
```

REMARK If you are using a mouse device, you must configure ZBasic for a mouse under "Configure". See "MSDOS Specific Configuration Options" for more information.

Also see MOUSE statement and MOUSE function in this appendix and in the main reference section.

DEF PAGE statement

FORMAT **DEF PAGE** *x1 , y1* **TO** *x2 , y2*

DEFINITION This statement defines the size of the screen used in print operations where the parameters are as follows:

x1,y1 = the upper left character position of the screen
x2,y2 = the lower right corner of the screen

ZBasic uses the screen size in scrolling the characters on the screen and in the CLS statement. Programmers may use this command as an aid in creating "WINDOWS".

EXAMPLE

```
REM Example of creating WINDOWS
MODE 7: COLOR ,255: REM CGA or EGA
:
"MENU BAR"
PRINT@(0,0)CHR$(2);"          MODE        FILES       EDIT"
PLOT 0,30 TO 1024,30
:
"WINDOW"
BOX 123,150 TO 910, 650          <--- Put a box around the window
PRINT @(33,4);"ZEDCOR WINDOWS"; <--- Window title
BOX 123,150 TO 910,121           <--- Put a box around the title bar
:
PRINT@(10,5);                    <--- Put cursor to first window position
DEF PAGE 10,5 TO 70,20
:
DO
  PRINT"HELLO THERE...";         <--- Text scrolls in window.
UNTIL LEN(INKEY$)
END
```

ZBasic keeps the text within the window specified.

This will force scrolling and CLS *n* to operate from the 5th row and 10th column to the 20th row and the 70th column. This will leave the last four rows (rows 21 to 24) and the last 9 columns (columns 71 to 79) unaffected by normal print operations.

REMARK This statement is most useful for displaying information and status on the screen that will not be erased by a CLS or by scrolling characters. The area outside the defined screen can be accessed using LOCATE or PRINT@ to locate the cursor in this area. Then normal printing can be done, except that none of the screen will be scrolled. When done printing in this area, a LOCATE or PRINT@ is again used to go back to the normal area (CLS will also home the cursor inside the normal area).

Use MODE to reset the screen to normal text coordinates.

MSDOS APPENDIX

DEF PAGE READ/WRITE statements

FORMAT

DEF PAGE **READ** [=] *expression*
DEF PAGE **WRITE** [=] *expression*

DEFINITION

ZBasic can access the extra pages of memory available in text modes on the IBM PC with EGA, CGA, MDPA and HERCULES graphic cards.

There are 4 pages of text when in 80 column text mode (ZBasic MODEs 2,3,4,6,10,11,12,14) and 8 pages of text when in 40 column text mode (ZBasic MODEs 0,1,8,9 [16 when 256K is on the EGA card]). EGA MODES 17-19 allow 1, 2 or 4 pages depending on whether there is 64, 128 or 256k on the board. See MODE for details.

ZBasic can write to and display any of the available screens. This allows the programmer to write an entire screen full of data while a different screen is being displayed; then display the new screen instantly. The syntax is as follows:

Set Display Page:
DEF PAGE READ [=] *expression*

Set Write Page:
DEF PAGE WRITE [=] *expression*

EXAMPLE

```
DEF PAGE WRITE = 1          <--- Writes to page 1 while page 0 is displayed.
CLS
FOR I=1 TO 20
  PRINT "STRING NUMBER";I
NEXT I
PRINT "Press a key..."      <--- You won't see this till you display page one
DEF PAGE WRITE=0            <--- Write back to page 0 to display message.
PRINT"Press a key..."
*DO: UNTIL LEN(INKEY$)
DEF PAGE READ=1             <--- Display page one.
*DO: UNTIL LEN(INKEY$)
DEF PAGE READ = 0           <--- Set display back to page zero.
```

This example will show you the results of "flipping" text pages.

REMARK

When the screen being displayed is not the same as the screen being written, the ZBasic "CLS" statement does not work. Also, the auto scrolling feature, which occurs when writing beyond the 25th row on the screen, does not work.

However, the ZBasic "CLS *expression* " statement will work regardless of the displayed page. Thus, use CLS 32 to clear on any page (32 is ASCII for " ").

Note: Hercules graphics support only page 0 and 1 (MODE 20).

DEF SEG statement

FORMAT **DEF SEG** [=] [*address*]

DEFINITION Defines the current segment in memory. The *address* is an integer number between 0 and 65,535 (or the signed integer numbers -32,768 to 32,767).

Any subsequent use of a BLOAD, BSAVE, CALL, PEEK or POKE definition specifies the offset into the segment (if the ZBasic segment option is not used in these commands).

EXAMPLE

```
DEFSEG =  &B000
BLOAD "TEXT.FIL" , 80
```

REMARK This will load the file `TEXT.FIL` into monochrome display memory (which starts at &B000) at an offset of 80 bytes.

See MEM, PEEK, POKE, BLOAD, BSAVE, VARPTR, VARSEG and the sections in this appendix entitled "Memory Considerations" and "MSDOS Memory Map" for more information.

DEF USR statement

FORMAT **DEF USR** *n* [=] *address* [, *segment*]

DEFINITION This statement is used to tell ZBasic where a user function is to be located in memory.

The difference with the regular ZBasic and the IBM version of DEF USR is in the definition of the address. *Address* is used as the offset into the segment given by *segment*. If the segment is not given, then ZBasic's code segment is used.

EXAMPLE

```
DEFUSR1 = LINE 100
```
This defines the subroutine at ZBasic's line 100 as user function 1.

```
DEFUSR1 = VARPTR(I),MEMD
```
This defines the address of variable I in ZBasic's data segment as user function 1. The subroutine must end with a far return.

```
DEFUSR1 = &H0, &H0B7D
```
This defines offset zero into the segment at &HB7D as address of user function 1. This is a very dangerous use of the USR function and is not recommended.

REMARK Use caution when specifying the segment. The subroutine must always be located at that specific address, which is very uncommon on the IBM machines.

Any subroutine that is called by specifying the segment must return from the subroutine with a far return. Otherwise, unpredictable results will occur.

Also see CALL, MACHLG and the section in the front of this manual "Machine Language".

END statement

FORMAT **END** [= *expression*]

DEFINITION The END statement is the normal way to exit a ZBasic program. On the MSDOS version, however, an error return code can be sent using the END = *expression.* This value can then be interrogated by the batch subcommands IF and ERRORLEVEL.

If the END = *expression* statement is not used to terminate a program, then the error code returned is zero.

ZBasic does two things depending on how END is used.

END USED WITH	RESULT
RUN	Control is returned to the Standard Line Editor.
RUN+ or RUN*	Control is returned to MSDOS.

EXAMPLE

```
END = 4
```

A program terminated by this statement could be tested by the following batch subcommand:

```
IF ERRORLEVEL 4  ECHO  ERROR IS AT LEAST 4
```

This subcommand will echo to the screen "`ERROR IS AT LEAST 4`".

REMARK See your MSDOS technical reference manual for more information on batch files and the ERRORLEVEL subcommand.

Also see SYSTEM in this appendix and STOP in the main reference section.

MSDOS APPENDIX

ERROR function

FORMAT **ERROR**

DEFINITION The same as the standard ZBasic ERROR function except that other errors are returned:

ERROR=11 An error was encountered with RMDIR, MKDIR or CHDIR.

Errors encountered depend on the statement but usually are related to that statement i.e. if you get an error 11 doing RMDIR either the directory didn't exist, the wrong diskette was used, a write protect was on the disk or the wrong pathname was used.

See ON COM ERROR GOSUB for errors returned from the communication ports.

EXAMPLE

```
ON ERROR GOSUB 65535
MKDIR "Mydir"
LONG IF ERROR>0
  LONG IF ERROR =11
    PRINT "Could not make a new directory!"
    PRINT "Please check your disk drive"
    INPUT "and press <ENTER> to continue";temp$
    GOTO "START"
  END IF
XELSE
  PRINT "A disk error occurred"
  PRINT ERRMSG$(ERROR)
  INPUT" <C>ontinue or <S>top";temp$
  IF temp$="C" THEN ERROR=0: GOTO "START"
  END
END IF
Program continues....
```

REMARK See PATH$, RMDIR, CHDIR, MKDIR, ERROR function, ON COM ERROR GOSUB, ON ERROR GOSUB, ERROR statement and the chapter "Trapping Disk Errors" in the main reference section.

FILES statement

FORMAT **FILES** [*filespec*]

DEFINITION Prints a directory of the files in the directory specified by filespec.

If filespec if not used the current directory contents are listed.

EXAMPLE

```
CLS
FILES "A:"
END
```

RUN

```
Fred         BAS       9843     03/23/87     9:45
Harry        BAS      23020     02/22/87    10:32
ZBasic       COM      92020     02/22/99    23:12
ZDEMO        COM      12312     12/23/86    12:34

  4 File(s)           13993949 bytes free
```

REMARK Also see RMDIR, CHDIR, MKDIR and your MSDOS reference manual for file specifications and pathname syntax. COMMAND.COM must be on the disk.

Note: Other ways to obtain directories during runtime:

To create a Directory file that you can read into your program (say into string arrays, so you can manipulate them) try using; SHELL"DIR>DIR.TXT". This creates a text file called "DIR.TXT" that will contain a listing of the current directory. This example program should give you some ideas:

```
CLS
DIM 80A$(100)
:
SHELL"DIR>DIR.TXT"
:
OPEN"I",1,"DIR.TXT"<--- This routine opens the file and loads it into A$(x).
DO
  READ#1, temp$;1
  IF ASC(temp$)=13 THEN X=X+1
UNTIL EOF(1) OR X>99
X=X+1
CLOSE#1
END
PRINT "There are";X;" items in this directory"
```

FIX command

FORMAT **FIX** (*spaces to indent*)

DEFINITION This command is used from the Standard Line Editor to remove line numbers (that are not referenced) and set the number of spaces to indent structures for the Full Screen Editor (not the line editor).

Unless this command or the INDENT command is used ZBasic will not normally indent in the full screen editor (it indents two spaces in the Standard Line Editor).

ZBasic structures are FOR-NEXT, DO-UNTIL, LONG-XELSE-END IF, WHILE-WEND, LONG FN-END FN and SELECT CASE, CASE ELSE, END SELECT.

This is the same as doing both UNNUM and INDENT.

EXAMPLE

```
LIST

10 DO
20   X=X+2
25   X=X-1
30 UNTIL X=100
40 PRINT"HELLO"
50 PRINT "This is just a test"
60 GOTO 10

FIX 10

<F10> <--- To go into the Full Screen Editor

10 DO
          X=X+2
          X=X-1
UNTIL X=100
PRINT "HELLO"
PRINT"This is just a test"
GOTO 10
```

REMARK Also see INDENT, UNNUM and RENUM*

FRE function

FORMAT **FRE** (*dummy argument*)

DEFINITION Returns the amount of free memory left in the system (divided by 16). Multiply the number returned by 16 to get the amount of free memory in bytes.

EXAMPLE

```
A# = FRE(x)
M# = A#*16          <---M# contains the amount of free  memory in bytes

RUN

345329
```

REMARK It is imperative that these variables be double precision to get the true amount of free memory.

Also see MEM and the sections in this appendix "Memory Considerations" and "MSDOS Memory Map".

Since there is no "Garbage Collection" with ZBasic, this function does not compact memory or return the remaining string space as BASICA does. String space is allocated at compile time.

INDENT command

FORMAT **INDENT** (*number of spaces to indent*)

DEFINITION This command is used from the Standard Line Editor to set the number of spaces to indent structures for the Full Screen Editor (not the line editor).

ZBasic normally does not indent in the full screen editor. It indents two spaces in the Standard Line Editor.

ZBasic structures are; FOR-NEXT, DO-UNTIL, LONG-XELSE-END IF, WHILE-WEND, LONG FN-END FN and SELECT CASE, CASE, CASE ELSE, END SELECT.

EXAMPLE

```
LIST

DO
  X=X+2
  X=X-1
UNTIL X=100
PRINT"HELLO"
PRINT "This is just a test"

INDENT 10

<F10> <--- To go into the Full Screen Editor

DO
         X=X+2
         X=X-1
UNTIL X=100
PRINT "HELLO"
PRINT"This is just a test"
```

REMARK Also see FIX, UNNUM and RENUM*

INKEY$ function
(Special checks for function and other keys)

FORMAT INKEY$

DEFINITION This function operates the same as described in the ZBasic reference section, except in its handling of function keys.

Normally, the INKEY$ function will return a string whose LEN=1 if a key is available, otherwise a null string is returned. However, on the IBM version only, the string returned will have a LEN=2 when a function key is pressed.

The first character in the string will be a null and the second character will be the value of the key pressed. See "ON INKEY$ statement" for the value corresponding to each function key.

EXAMPLE

```
DATA F1,F2,F3,F4,F5,F6,F7,F8,F9,F10
DATA HOME, UP, PAGE UP, NONE, CURSOR LEFT, NONE, CURSOR RIGHT
DATA NONE, END, CURSOR DOWN, PAGE DOWN, INSERT, DELETE
:
CLS:PRINT"Key";TAB(30);"ASCII Code of 2nd char."
:
DO
  DO
    A$=INKEY$
  UNTIL LEN(A$)
  LONG IF LEN(A$)=2
    B$=RIGHT$(A$,1)
    LONG IF (ASC(B$)>58) AND (ASC(B$)<69)
      RESTORE ASC(B$)-59
      READ C$
      PRINT "You pressed: ";C$;TAB(30);ASC(B$)
    ELSE IF
      RESTORE 10+ASC(B$)-71
      READ C$
      PRINT "You pressed: ";C$;TAB(30);ASC(B$)
    END IF
  END IF
UNTIL LEN (A$)=1
END
```

This example will print the function key you pressed if one is detected on the INKEY$ function.

REMARK See ON INKEY$ and INKEY$ statement in this section for more information on how to make full use of the function keys.

MSDOS APPENDIX

INKEY$ statement

FORMAT **INKEY$** (*expression*)

DEFINITION This statement is used to enable or disable function key interrupts.

The function keys can be used to control program flow with ON INKEY$ GOTO (see ON INKEY$ statement in elsewhere in this appendix).

The *expression* in the INKEY$ statement does the following:

zero ---> disables the function keys
non-zero ---> enables the function keys

Enabling or disabling the function keys does not destroy the previous ON INKEY$ key definitions, it simply decides whether or not ZBasic should check for function keys.

EXAMPLE

```
ON INKEY$(1) GOTO "F1"
ON INKEY$(2) GOTO "F2"
:
INKEY$(1) <--- Function key interrupts ON
:
"Event Loop"
I$=INKEY$:IF I$="S" THEN STOP
GOTO "Event Loop"
:
INKEY$(0) <--- Function key interrupts OFF
:
"F1"
PRINT "F1":GOTO "Event Loop"
"F2"
PRINT "F2":GOTO "Event Loop"
END
```

REMARK The ability to turn the function keys on and off is very useful when parts of a program use function keys and other parts do not.

If a subroutine does not want to use function keys, the INKEY$(0) statement is used at the beginning and then INKEY$(1) is used when the routine is done.

Without this statement, ON INKEY$(n) RETURN would have to be done to all function keys at the beginning and ON INKEY$(n) GOTO line# at the end.

KEY command

FORMAT

KEY ON
KEY OFF
KEY LIST

DEFINITION

Controls the display of function key options from the standard Line Editor.

KEY ON — Shows the function key equivalents on the last row of the screen. This is the default.

KEY OFF — Hides the function key equivalents.

KEY LIST — Prints a list of the function key equvalents:

EXAMPLE

```
KEY  LIST

F1        LIST
F2        RUN
F3        LOAD
F4        SAVE
F5        FIND
F6        EDIT
F7        CONFIG
F8        COMPILE
F9        KEY
F10       EDITOR
```

REMARK

Also see INKEY$ statement and ON INKEY$ in this appendix for other ways of controlling function keys during runtime.

Note: The keys on the bottom of the screen for the Full Screen Editor are different. See "Full Screen Editor" at the end of this appendix for specifics.

Note: KEY ON and KEY OFF are ignored during runtime.

MSDOS APPENDIX

LOCATE statement

FORMAT **LOCATE** [*X* , *Y*] [,[*cursor on/off*] [, [*start line*] [, *stop line*]]]

DEFINITION This statement handles all of the cursor functions.

Parameter	Definition
X,Y	horiz, vert coordinate on screen.
on/off	0= cursor off, not 0=cursor on
start line	start line for cursor character 0-13 (cursor appearance)
stop line	stop line for cursor character 0-13

The start and stop lines for the cursor determine the size and vertical position of the cursor; `LOCATE ,,,0,13` makes a fat cursor. `LOCATE ,,,13,13` makes a thin cursor. This example illustrates the different cursor types:

```
DO
  INPUT"Start line, Stop line",x,y
  LOCATE ,,, x,y
UNTIL X<0 OR X>13
```

For the monochrome adapter, the values of these two parameters can be from 0 to 13. Zero specifies the top of the character block and 13 specifies the bottom. With the graphics adapter, 0 is top and 7 is the bottom. The cursor can be turned on and off without the start and stop lines being affected.

EXAMPLE

LOCATE 0,20,0
This sets the cursor location at column zero of row 20. The cursor is also turned off.

LOCATE ,,1,3,4
This turns the cursor back on and sets the cursor start and stop lines to 3 and 4 (which is about the middle of the character on the graphics adapter).

REMARK Note: The X,Y orientation can be reconfigured to be Y,X (row,column). See "MSDOS Specific Configuration Options" in this appendix for details.

Also note that ZBasic defaults to accessing character cells using numbers from 0 to 79 accross and 0 to 24 down. BASICA LOCATE uses 1 to 80 and 1 to 25 respectively. You may configure ZBasic for this under "Configure".

MEM function

FORMAT

MEM
MEM *letter*
MEM STR
MEM BCD
MEM ARR *array variable name* (*dummy expression*)

DEFINITION

This function returns the segment address of specific portions of a ZBasic compiled program.

The segments returned are as follows:

MEM	Memory remaining in INDEX$ segment
MEMC	CODE segment
MEMD	DATA segment
MEME	EXTRA segment
MEMS	STACK segment
MEMI	INDEX$ segment
MEM STR	Simple strings
MEM BCD	BCD variables.
MEM ARR A$(1)	Array A$ begins.

EXAMPLE

```
CLEAR 2000: REM  Clear room for INDEX$
DIM A$(100), A#(200),A!(300)
:
PRINT MEM, MEMC, MEMD, MEME, MEMS, MEMI
PRINT MEM STR
PRINT MEM BCD
PRINT MEM ARR A$(0)
PRINT MEM ARR A#(200)
PRINT MEM ARR A!(300)
```

REMARK

When using the MEM ARR function the string array is given only one dummy argument.

For example:

```
DIM A$(10,10)
A = MEMARR A$(12)
```

Only one argument is given, even though the array is 2 dimensional. It does not matter what the value inside the ( ) is , it is only a dummy argument. It's 12 in this example but could have been any number. In effect, it is the same as VARPTR A$(0,0) and reading the segment from the variable VARSEG. See VARPTR for further info.

Also see VARPTR, VARSEG and the chapter in this appendix entitled "Memory Considerations" and especially the MSDOS Memory Map.

MEM command

FORMAT **MEM**

DEFINITION The MEM command now gives some extra information about the memory usage of variables. The following is an example of its output:

EXAMPLE

```
ZBasic Ready

COMPILE

ZBasic Ready

MEM

00167   Text
64238   Memory
14364   Object
01232   Buffer + Integer size
00008   BCD size
00512   String size
00001k  BCD/String size
00392k  Available

ZBasic Ready
```

REMARK Do not confuse the MEM command with the MEM function. The MEM command is not a runtime feature.

The above values for Object, Buffer+Integer size, BCD size, String Size, and BCD/STRING (array) size will be true only after a successful COMPILE, RUN, RUN* or RUN+.

MKDIR statement

FORMAT **MKDIR** pathname

DEFINITION Creates a new directory (same as the DOS MKDIR command).

EXAMPLE

```
A$=PATH$
MKDIR "Mydirect"
IF ERROR=11 THEN PRINT"Disk error making directory": ERROR=0
END
```

REMARK Also see PATH$, CHDIR and RMDIR.

Note: If an error is encountered when doing CHDIR, RMDIR or MKDIR ZBasic returns an error eleven (11) in the ERROR function. See ERROR function, ERROR statement, ON ERROR GOSUB and the chapter "Disk Error Trapping" in the front of this manual.

MSDOS APPENDIX

MODE statement

FORMAT **MODE** *expression*

DEFINITION MODE sets the system screen attributes. MODE can be set in the range 0-20 and takes advantage of all popular graphic and text devices available for IBM PC's and compatibles.

Color Graphics Adaptor (CGA)

MODE 0-7 Same as the standard ZBasic modes defined in the reference section. All of these modes will use the color of the CGA card (or EGA card when dip switches set to CGA mode) if it is available.

This card uses memory located at &HB800 and uses between 2,048 and 16,384 bytes depending on the MODE. Allows 16 colors in text and low resolution graphics modes only. Four colors in MODE 5 and 2 colors in MODE 7.

Monochrome Display and Printer Adaptor (MDPA)

MODE 2 If a Monochrome Display and Printer Adaptor is all that is available then only MODE 2 and 3 will be allowed. This board uses memory at adress &HB000. It uses 4,096 bytes.

CGA Black and White

MODE 8-15 On the IBM, Modes 8-15 are the same as 0-7 with the exception that only black & white is used on the screen. These modes can be slightly faster, especially in graphics.

Enhanced Graphics Adpator (EGA)

MODE 16-19 These modes require the Enhanced Graphics Adaptor (EGA). Resolution varies depending on the amount of memory on the board (see MODE chart). This board uses memory at &HA000 and uses 8,192 bytes for MODE 16 and 32,768 bytes for MODE 17, 18 and 19 (in monochrome graphics modes colors are: 0=black, 1=low-intensity, 2=blinking, 3=high-intensity).

Hercules and Hercules Plus Monochrome Graphics Cards

MODE 20
MODE 2 (text only) This mode allows the use of the Hercules or Hercules Plus high resolution graphics boards (or compatibles). Text and graphics may be intermixed if the Hercules character driver; `HERC.BIN` has been loaded first. No other graphic modes will work with the Hercules cards. It uses memory at &H8000 and uses 16,384 bytes. To utilize high-resolution text-only, use MODE 2.

Note: Also see SCREEN, COLOR, PALETTE, DEF PAGE and the section in the main reference "Graphics".

continued...

MSDOS VERSION MODE CHART

MODE Number	SCREEN Number	TEXT Column/Row	GRAPHICS horizon x vert	MAX PAGES	COLORS Available	MONITOR TYPE
0		40x25	character	8	16	C/RGB
1		40x25	40x25	8	16	C/RGB
2		80x25	character	4	16	MONO
3	0	80x25	80x25	4	16	C/RGB
4		80x25	character	4	16	C/RGB
5	1	40x25	320x200***	8	4	C/RGB
6		80x25	character	4	16	C/RGB
7	2	80x25	640x200***	4	2	C/RGB
8		40x25	character	8	B/W	C/RGB
9		40x25	40x25	8	B/W	C/RGB
10		80x25	character	4	B/W	MONO
11		80x25	80x25	4	B/W	C/RGB
12		80x25	character	4	B/W	C/RGB
13		40x25	320x200***	8	B/W	C/RGB
14		80x25	character	4	B/W	C/RGB
15		80x25	640x200***	4	B/W	C/RGB
16	7	40x25	300x200•	2/4/8*	16	C/RGB
17	8	80x25	640x200•	1/2/4	16	C/RGB
18	10	80x25	640x350•	1/2	3	MONO
19	9	80x25	640x350•	1/2	4/16**	C/RGB
20		80x25	720x348••	2	2	MONO

Screen Number is the corresponding QuickBASIC equivalent and may be used instead of MODE. See SCREEN.
*The maximum number of pages for modes 16-19 depends on the total graphic memory installed. The first number is 64K, the second 128K and the last number is 256K. See DEF PAGE READ/WRITE. ** Mode 19 requires 128K or more to display 16 colors. C/RGB stands for Color or RGB monitor. MONO stands for monochrome monitor.
***Requires CGA or EGA. • Requires EGA. •• Requires Hercules or Hercules+ display or compatible.

REMARK To determine what board is installed on a computer see CARDTYPE function in this appendix. Also see SCREEN, DEF PAGE READ/WRITE and COLOR in this appendix for information on setting text types and colors and manipulating pages.

Also see PALETTE.

MOUSE function

FORMAT **MOUSE** (*expression*)

DEFINITION This function returns information from the current mouse driver as defined by the DEF MOUSE statement. *Expression* will determine the value returned as follows:

MOUSE(0) ---> Resets and returns true if installed.
MOUSE(1) ---> Returns X position
MOUSE(2) ---> Returns Y position
MOUSE(3) ---> Returns button status

The X and Y positions returned are in terms of ZBasic's graphic coordinate system.

If *expression* is not 0 to 3, then zero will be returned. Also, MOUSE(0) is supported only for the mouse driver (i.e. for DEF MOUSE = 0).

EXAMPLE

```
DO
  PLOT MOUSE(1), MOUSE(2)
UNTIL MOUSE(3)
```

This will plot on the screen the position of the mouse device until a button is pushed.

REMARK The MOUSE function does not operate exactly the same for all DEF MOUSE types. Use the following for more specific information:

DEF MOUSE = 0 ---- MOUSE DRIVER

MOUSE(0) Resets the mouse hardware and software and returns 0 (false) if hardware is not installed, otherwise returns -1 (true). ZBasic always initially resets the mouse.

MOUSE(1) Returns the horizontal position of the mouse.

MOUSE(2) Returns the vertical position of the mouse.

MOUSE(3) Returns button status from 0 to 3. Zero if both buttons up, 1 or 2 if only one buttom is pushed and 3 if both buttons down.

The mouse cursor can also be shown and hidden by using the MOUSE(4) and MOUSE(5) statements. See MOUSE Statement.

Note that the ZBasic has to be configured to support the mouse. See configuration and DEF MOUSE.

continued...

MOUSE function continued

DEF MOUSE = 1 or 2 ---- JOYSTICK DRIVER

MOUSE(1) Returns horizontal position of joystick

MOUSE(2) Returns vertical position of joystick

MOUSE(3) Returns button status from 0 to 3. Zero if both buttons up and 3 if both buttons down. ZBasic debounces the joystick buttons for 1 millisecond.

DEF MOUSE = 3 ---- LIGHT PEN DRIVER

MOUSE(1) Returns last horizontal position. If pen switch is currently down, the X and Y positions are updated and the new X position is returned; else the last position is returned.

MOUSE(2) Returns last vertical position, operating the same as MOUSE(1).

MOUSE(3) Returns pen switch status from 0 to 1 (0 if light pen switch not down/not triggered, 1 if down/triggered). If pen switch is down, the X and Y positions are updated.

If a mouse driver is installed, the light pen driver will no longer work. The MOUSE function will then return the mouse position and buttons instead of the light pen, still updating position only when a button is pressed.

MOUSE statement

FORMAT **MOUSE** (*expression*)

DEFINITION This statement is used to show and hide the mouse cursor.

It is only used for the mouse driver (i.e. DEF MOUSE = 0; see "DEF MOUSE Statement" in this appendix).

The *expression* defines the operation as follows:

MOUSE(4)	Show mouse cursor
MOUSE(5)	Hide mouse cursor

If *expression* is not 4 or 5, the statement is ignored.

EXAMPLE

```
X=1
MODE 7
I=MOUSE(0)                              <--- Reset mouse hardware
MOUSE(4)                                <--- Let's see that mouse cursor
:
"Mouse"
DO
  IF LEN(INKEY$) THEN STOP              <--- Press a key to stop
  PLOT MOUSE(1), MOUSE(2)
UNTIL MOUSE(3) <---Press mouse button to toggle Mouse cursor
:
X=X*-1                                  <--- Toggle mouse cursor
IF X>0 THEN MOUSE(4) ELSE MOUSE(5)
GOTO "Mouse"
```

REMARK It is important to note that the number of calls to one of the mouse statements must be equal to the number of calls to the other to get the cursor to the same state. For example, if MOUSE(5) is called 10 times to hide the cursor, then MOUSE(4) must be called 10 times to show the cursor.

Also, ZBasic initially resets the mouse and leaves the cursor off (same as the MOUSE(0) function); one call to MOUSE(4) will turn the cursor on.

This cursor can only be seen in ZBasic MODE's 5,7,13,15, 16.

ON COM ERROR GOSUB statement

FORMAT **ON COM ERROR GOSUB** *line*

DEFINITION Used to call an error routine when an error is detected while reading data from the communications ports.

EXAMPLE

```
OPEN"C", -1, 1200,,,,,,20000
ON COM ERROR GOSUB "COM Error Routine"
.
.
.
"COM Error Routine"
PRINT"A Communications error has occurred!"
PRINT"The error is: ";
X=USR5(-1)            <--- Change to -2 if other port is being used
SELECT CASE X
  CASE AND 2^9        <--- See USR5 for errors associated with bits 9-12
    PRINT"Overrun Error"
  CASE AND 2^10
    PRINT"Parity Error"
  CASE AND 2^11
    PRINT"Framing Error"
  CASE AND 2^12
    PRINT"Break detect"
  CASE ELSE
    PRINT"UnKnown Error!"
END SELECT
INPUT"<C>ontinue or <S>top?";temp$
temp$=UCASE$(temp$)
IF temp$="C" THEN RETURN
END
```

REMARK See COM BUFF, COM ON, COM OFF, OPEN "C" and USR5(-1) in this appendix for details. Also see OPEN"C" in the main reference section for other information.

ON INKEY$ statement

FORMAT

ON INKEY$(*expression* **) GOTO** *line#*
ON INKEY$(*expression* **) RETURN**

DEFINITION

This statement is used to control the action when a function key is pressed. ZBasic supports 20 of the function keys on the IBM standard keyboard. Table 1 on the next page shows the function keys and the corresponding number associated with each.

When using the ON INKEY$ statement, *expression* determines which function key is being defined according to Table 1. The function key is not actually recognized until a ZBasic keyboard function is implemented, such as INPUT, LINE INPUT, and INKEY$ function.

When the GOTO is used, the *line#* specifies where the program will continue execution after the function key is hit. When RETURN is used, the function key is no longer implemented.

All function keys are disabled until the INKEY$() statement is used. See "INKEY$() statement" in this appendix for more information.

EXAMPLE

```
ON INKEY$(1) GOTO "F1"
ON INKEY$(2) GOTO "F2"
:
INKEY$(1)
"Event Loop"
I$=INKEY$:IF I$="S" THEN STOP
GOTO "Event Loop"
INKEY$(0)
:
"F1"
PRINT "F1":GOTO "Event Loop"
"F2"
PRINT "F2":GOTO "Event Loop"
END
```

REMARK

Remember to use the INKEY$(1) statement to enable the function keys; otherwise, ZBasic doesn't check to see if the ON INKEY$() statement was used. See "INKEY$()" statement in this appendix.

continued...

ON INKEY$ statement continued

When a GOTO is made from a function key, the current program execution is terminated and then restarted at the location specified in the ON INKEY$() GOTO statement. Thus, this program location cannot be nested in a subroutine! If a RETURN is executed before a GOSUB, the program will stop and ZBasic will return to the editor (or the operating system if in a compiled program).

The following is a list of the function keys supported by ZBasic and the numbers with which each key is associated. The ON INKEY$ column refers to the number of each key in the ON INKEY$(*expression*) statement. The INKEY$ column is the value that the INKEY$ function will return when the function key is not implemented (see INKEY$ function in the this appendix).

TABLE 4: Function Key Codes

KEY	ON INKEY$	INKEY$
F1	1	59
F2	2	60
F3	3	61
F4	4	62
F5	5	63
F6	6	64
F7	7	65
F8	8	66
F9	9	67
F10	10	68
HOME	13	71
CURSOR UP	14	72
PAGE UP	15	73
CURSOR LEFT	17	75
CURSOR RIGHT	19	77
END	21	79
CURSOR DOWN	22	80
PAGE DOWN	23	81
INSERT	24	82
DELETE	25	83

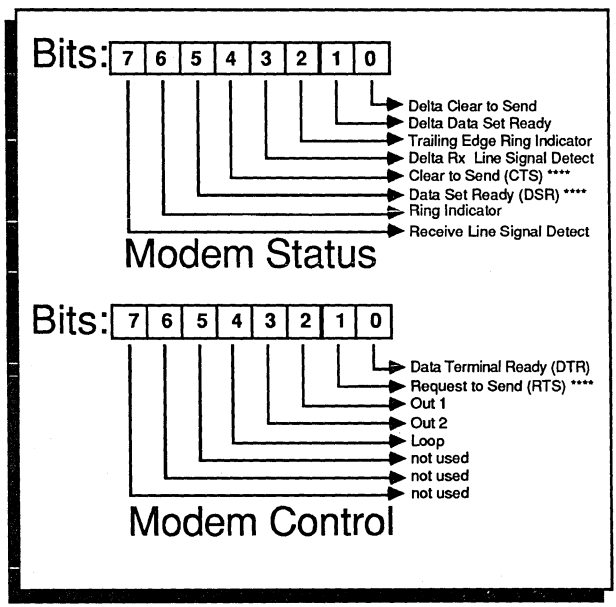
MSDOS APPENDIX

OPEN "C" statement

FORMAT **OPEN "C",{ -1 | -2} [,[*baud*][,[*parity*][,[*stopbit*][,[*word length*][,[*status*][,[*control*][,buffer size]]]]]]]**

DEFINITION The OPEN "C" statement has three additional parameters more than is provided with the Standard ZBasic.

•*Status* and •*Control*; can be used to control the handshaking on the RS-232 cable when writing to the port - *modem status* and *modem control.* The modem status and modem control parameters indicate the following:



The default bits for these two parameters are shown by the four asterisks (****) after the bits set. This makes the default values &X00110000 (or 48 decimal) for modem status and &X00000010 (or 2 decimal) for modem control.

• *buffer size* is used for loading incoming data with COM ON and COM OFF. COM ON is automatically executed when OPEN"C" is used.

The buffer defaults to 256 bytes but may be configured up to 32,700 bytes. Also see ON COM ERROR GOSUB and COM BUFF.

continued...

OPEN"C" statement continued

EXAMPLE

```
OPEN "C",-1,300,,,,0
```

This will force ZBasic to ignore the signal lines DSR and CTS when writing to the port. This will normally work at 300 baud.

REMARK

It is important to note that all of these optional parameters affect *both* COM1 and COM2.

When a character is written to the COM port, ZBasic does the following:

1) Sends an indicator to the modem control register using the value given in the modem control parameter. This is usually a Request to Send (RTS).

2) Waits for the appropriate signals from the modem status register given in the modem status parameter. These are usually Data Set Ready (DSR) and Clear to Send (CTS).

3) Waits for the transmitter holding register to be empty and then sends the character to the port.

If the default parameters do not work properly on your machine, try setting the Data Set Ready and/or Clear to Send bits to zero and/or turning the Data Terminal Ready bit to one.

Also see COM ON, COM OFF, COM BUFF, ON COM ERROR GOSUB, USR5 and the chapter in this section call "RS-232 Communications". Also see OPEN"C" in the main reference section for other information.

PAGE LPRINT statement (Screen Dump)

FORMAT **PAGE LPRINT**

DEFINITION A screen dump to the printer will occur if the PAGE LPRINT statement is executed. This statement is the same function as typing "Shift-PrtSc" from the IBM keyboard.

EXAMPLE

```
PRINT"HELLO"
PAGE LPRINT
END
```

This will cause the entire screen image to be dumped to the printer.

REMARK This statement is most useful for printing screen graphics created by ZBasic. Graphics are not normally dumped to the printer, however. The program "GRAPHICS.COM" that comes with MSDOS must be run before using the program in order to install the graphics printer driver.

See the MSDOS reference manual for more information.

Requires IBM PC compatible screen dump routines. May not function on not-so-compatibles.

PAINT statement

FORMAT **PAINT** (*x,y*)

DEFINITION Fills a section of the screen with the current color. Same as the ZBasic FILL statement.

PAINT will use the default coordinate system of 1024x768. If you wish to use pixel coordinates, or your own coordinate system, use the COORDINATE ststement in the beginning of your program (see coordinate in the main reference section).

EXAMPLE

```
CIRCLE  512, 383, 300
COLOR 3
PAINT (512,383)
END
```

REMARK See FILL, GET, PUT, PLOT, BOX FILL, CIRCLE FILL, PALETTE, MODE, COLOR (in this appendix and in the front reference section), COORDINATE WINDOW and the section in the front of this manual entitled "Graphics".

Note: BOX FILL and CIRCLE FILL are much faster than PAINT or FILL.

In some modes COLOR, attribute can be used to set the color of the background. This is also much faster than FILL or PAINT when the entire background needs to changed.

PALETTE statement (EGA only)

FORMAT **PALETTE** *attribute , color*

DEFINITION Changes a color in the EGA color palette. This statement will only work with PC's equipped with an EGA card.

attribute	0-15
color	0-63

EXAMPLE

```
COLOR = 1
CIRCLE 100,100,50: REM draw a blue circle
DELAY 2000:
PALETTE 1,2
:
REM the circle will instantly change to green and all
REM subsequent writes with color =1 will show
REM as green.
```

REMARK This statement allows the user to select a palette of 16 colors out of 63 available colors.

The PALETTE statement works only on systems with the Enhanced Graphics Adaptor (EGA).

Each attribute is paired with an actual display color.

PATH$ function

FORMAT **PATH$** (drive number)

DEFINITION The first format returns a string containing the current path of the specified drive.

drive number=0	default drive,
drive number=1	drive A:
drive number=2	drive B:
drive number=3	drive C:
drive number=4	drive E:
	etc.

EXAMPLE

```
A$ = PATH$ (0)
PRINT "Current Path = ";A$

RUN

Current Path =  C:\ZBASIC\TEMP
```

REMARK This can be used to save the current directory, so that the programmer can change to other directories (with the CHDIR statement) and have a way of returning to the original directory. See also CHDIR. See PATH in the main reference section.

MSDOS APPENDIX

PEEK function

FORMAT **PEEK** [WORD] (*address* [, *segment*])

DEFINITION This statement is used to read a particular address in memory. The address is actually the offset into the segment given by *segment.* If the *segment* parameter is not given, then the data segment used by ZBasic will be used as the segment.

EXAMPLE

```
PEEK WORD (&H00CC, 0)
```
Returns the mouse interrupt vector.

```
PEEK (0, &HB800)
```
Returns the first location on the screen.

```
PEEK WORD (VARPTR(I))
```
Returns the value of the variable I.

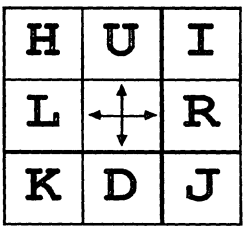
REMARK By specifying the segment, every address available on the IBM PC can be accessed. The PEEK is done much faster, however, when the segment is not given.

This statement is most frequently used in directly accessing screen memory (although IBM does not recommend doing this). For this purpose, use &HB000 for the segment if you have the monochrome adapter, and &HB800 if you have the color graphics adapter or EGA in CGA modes.

PLOT USING statement

FORMAT **PLOT USING** *X* , *Y* , *string* [, *magnification*]

DEFINITION This statement is used to plot a set of pixels on the screen in a pattern defined by *string* starting at the location *X,Y*. The starting location *X,Y* defines a point on the screen according to the ZBasic graphic coordinate system. The simple string *string* tells ZBasic where to plot each point corresponding to the one before it. The following characters are accepted: "UDLRHIJK", which control direction, and "+" or "-", which turn plotting on and off. The letters specify direction as follows:



EXAMPLE

```
PLOT USING 512,383,"UUUURRRDDDDLL"
```

This example plots a rectangle in the middle of the screen.

```
PLOT USING 512,383,"UUUU-RRR+DDDD",2
```

This example just plots the vertical halves of the rectangle in the previous example and twice as big.

```
MODE 7                          <--- Change MODE for your system.
A$="UUURRRRRRDDDLLLLLL"
FOR X= 1 TO 90 STEP 5
  COLOR=-1
  PLOT USING 0, 767, A$,X
  COLOR=0
  PLOT USING 0, 767, A$,X
NEXT
END
```

REMARK The PLOT USING statement only works in ZBasic graphic MODEs 5, 7, 13, 15, 16, 17, 18, 19 and 20 unless ZBasic is configured to not have IBM compatible graphics, in which case the PLOT USING function is completely disabled.

Each pixel is plotted in the color last set by the COLOR statement; thus, a pattern can be erased by setting COLOR = 0 and replotting.

When turning the plotting back on with a "+" imbedded in the string, note that the pixel at that point is plotted.

Note: This statement is similar to the DRAW statement found in BASICA.

POKE statement

FORMAT **POKE** [WORD] *address* , *data* [, ***segment***]

DEFINITION This statement is used to set a particular address in memory to a value determined by *data.*

The address is actually the offset into the segment given by *segment.* If the *segment* parameter is not given, then the data segment used by ZBasic will be used as the segment.

EXAMPLE

```
POKE WORD &H00CC,LINE 10,0
```
Sets the mouse interrupt vector to line 10 (not recommended!).

```
POKE 0,ASC("A"),&HB800
```
Sets the first location on the screen to "A".

```
POKE WORD VARPTR(I%),0
```
Sets variable I% to 0.

REMARK By specifying the segment, every address available on the IBM PC can be accessed. The POKE is done much faster, however, when the segment is not given.

This statement is most frequently used in writing directly to screen memory (although IBM does not recommend doing this). For this purpose, use &HB000 for the segment if you have the monochrome adapter, and &HB800 if you have the color graphics adapter or using the EGA card in CGA mode.

RENUM* command

FORMAT **RENUM*** *new, old , increment*

DEFINITION Adds line numbers to programs without line numbers. The compliment of UNNUM.

EXAMPLE

```
FOR X=1 TO 100
  PRINT X
NEXT
END

RENUM*

LIST

10 FOR X=1 TO 100
20   PRINT X
30 NEXT
40 END
```

REMARK Also see UNNUM, FIX and INDENT in this appendix and RENUM in the main reference section.

RMDIR statement

FORMAT **RMDIR** *pathname*

DEFINITION Removes a directory. Only empty directories may be removed.

EXAMPLE

```
RMDIR "Mydirect"
IF ERROR=11 THEN PRINT"Does not exist!":ERROR =0
```

REMARK Also see PATH$, CHDIR and RMDIR.

Note: If an error is encountered when doing CHDIR, RMDIR or MKDIR ZBasic returns an error eleven (11) in the ERROR function. See ERROR function, ERROR statement, ON ERROR GOSUB and the chapter "Disk Error Trapping" in the front of this manual.

SCREEN function

FORMAT SCREEN (*row*, *column* [, *color*])

DEFINITION If *color* is zero or is not used this function returns the ASCII code for the character on the active screen at the specified row and column. Only valid in text modes.

If *color* is used and is non-zero the color of the character at the screen location specified by row and column is returned.

row expression from 0-24.

column expression from 0 - 39 or 0 - 79, depending on current mode.

color an expression that evalutes to a true (not zero) or false(zero)

The upper left corner is defined the same way as the LOCATE function (see "Special MSDOS Configuration Options" in this appendix).

The number returned when using the color parameter may be interpreted as follows:

```
(number MOD 16)                          ; foreground color
( ( ( number - forground ) / 16 ) MOD 128 )   ; background color
```

EXAMPLE

```
A = SCREEN (9,9)
REM If character at position 9,9 is "B", A will equal 66
:
A = SCREEN ( 2,2,1)
REM The variable A will equal the color attribute of character at position 2,2
```

REMARK Also see COLOR, SCREEN statement and MODE.

SCREEN statement

FORMAT **SCREEN** *mode number*

DEFINITION Allows changing from one graphic type to another. Similar to ZBasic's MODE statement (added for compatibility to other BASIC languages).

mode number 0,1,2,7,8,9,10.

ZBasic MODE	SCREEN *mode number* equivalent
2	0
5	1
7	2
16	7
17	8
18	10
19	9

EXAMPLE

```
CLS
SCREEN 0
PRINT "HELLO"
END
```

Prints "HELLO" in monochrome mode. Same as MODE 2 using ZBasic MODE statement.

REMARK See MODE, PALETTE, VIEW SCREEN, DEF PAGE READ, DEF PAGE WRITE, SCREEN function, COLOR and the chapter on graphics in this section.

SHELL statement (CALL)

FORMAT **SHELL** *string*

DEFINITION The SHELL statement followed by *string* will load and execute another program specified by *string*. If a null (empty) string is specified, then MSDOS will be loaded and executed, in which case typing EXIT in DOS will return to ZBasic.

This is identical to the old ZBasic CALL *string* statement.

String must be either a string variable or a quoted string.

EXAMPLE

```
SHELL "DISKCOPY A: B:"
```
This will perform a diskcopy as if it was typed in from the DOS command line.

```
SHELL ""
```
This will go directly into DOS and give the DOS prompt.

```
SHELL "ZBASIC"
```
This will actually load and execute ZBasic. Typing QUIT will then return to the original ZBasic.

```
A$ = "DIR A:*.BAS": SHELL A$
```
This will get the directory of all .BAS files on the A drive.

REMARK This statement can be very useful for executing batch files -- just use the name of the .BAT file (batch file) for *string*.

There must be at least 17k of memory free to use the SHELL statement.

If COMMAND.COM is not found, the message "File not found" will be echoed to the display and control will be returned to ZBasic.

TFORMAT statement

(Hercules® or Hercules Plus® graphic boards only)

FORMAT **TFORMAT** [=] *expression*

DEFINITION The TFORMAT statement is used to set the text format when using text in the Hercules graphics MODE 20:

The values for *expression*:

0	Reverse video
1	normal (PSET); Overlays graphics
2	XOR mode. XOR text over background.

EXAMPLE

```
' Make sure ZBasic (or your program) was loaded using the
' ZHERC.BAT batch file (or your own batch file) so that the
' HERC.COM text driver is loaded into memory first.
:
MODE 20: REM Hercules graphics mode ONLY!
:
TFORMAT=1       <--- Regular text
PRINT"HELLO"
:
TFORMAT=0       <--- Reverse text
PRINT"HELLO"
DELAY 2000
MODE 2          <--- Back to MODE 2 for editing
END

RUN

HELLO
HELLO <--- Prints in black on white
```

REMARK Also see MODE, and the section about "Hercules and Hercules Plus Graphics" in this appendix.

TIME$ statement

FORMAT **TIME$** = [*hour*] [, [*minute*] [,*second*]]

DEFINITION This statement is used to set the current time. Any of the three parameters can be omitted, in which case the parameter will not be changed. The following values are accepted:

hour:	0 - 23
minute:	0 - 59
second:	0 - 59

EXAMPLE

```
PRINT TIME$
TIME$ = 17,0,0
PRINT TIME$
END
```

RUN

```
16:33:12
17:00:00
```

REMARK If any of the parameters used are not in the accepted range, the current time will not be changed.

```
TIME$ = ,,0
```
This only sets the seconds to zero, not destroying the current hours and minutes.

See TIME$ function in the main reference section.

TIMER function

FORMAT TIMER

DEFINITION Returns the number of seconds elapsed since midnight.

EXAMPLE

```
START# = TIMER
:
FOR X = 1 TO 32000:NEXT
:
FINISH# = TIMER
:
T#= FINISH# - START#
PRINT "The loop took ";T#;" seconds"

RUN

The loop took 2 seconds
```

REMARK Since the number of seconds elapsed since midnight can be greater than 65535, the variable must be BCD. The TIMER function can also be used to re-seed the random number generator when used with the RANDOMIZE statement.

UNNUM command

FORMAT UNNUM

DEFINITION Removes line numbers from lines that are not referenced elsewhere in the program by a GOSUB line number, GOTO line number etc.

Extremely useful cosmetic command for removing unsightly line numbers from your old BASIC programs (we should have used this sentence in our advertisements).

EXAMPLE

```
10 FOR X=1 TO 100
20   PRINT X
30 NEXT
35 GOTO 10
40 END

UNNUM

LIST

10 FOR X=1 TO 100
     PRINT X
   NEXT
   GOTO 10
   END
```

REMARK Also see RENUM* in this appendix and FIX and INDENT.

MSDOS APPENDIX

USR1 function (Check End Of File status)

FORMAT **USR1**(*filenumber*)

DEFINITION USR1 is a *predefined* user function available on the IBM PC. This user function is equivalent to the EOF(filenumber) function. The result is -1 if end of file, 0 otherwise.

Use the EOF function instead. This is retained only for compatibility with older versions of ZBasic. See EOF in the main reference section of this manual.

USR2 statement (Set the the clock constant)

FORMAT **USR2** (*expression*)

DEFINITION USR2 is a *predefined* user function available on the IBM PC version. USR2 is used to control the 1millisecond time constant used in the DELAY statement.

The ZBasic DELAY statement should delay a specified number of milliseconds (1/1000 of a second). This delay time is, however, very dependent on the actual speed of the computer. The delay time constant defaults to a 1ms delay on the IBM PC (i.e. 4.77 megahertz clock speed using the 8088 microprocessor).

If using ZBasic on a different speed computer, then use the USR2 statement to adjust the time constant.

IBM PC	--->	expression = 300
faster computer	--->	use a larger expression
slower computer	--->	use a smaller expression

This delay time constant is also used in the SOUND statement to specify the duration.

EXAMPLE

```
USR2(470)
```

This will set the time constant to 470. This is the value used on the IBM PC/AT to correct the delay times.

REMARK This time constant can also be altered in configuration, which would change the default value. See "MSDOS Specific Configuration Options" in this appendix.

USR3 function (Check keyboard status)

FORMAT USR3(*expression*)

DEFINITION USR3 is a *predefined* user function available on the IBM PC. This user function is used to control keyboard input and status as follows:

USR3(0)---> Returns the next character struck from the keyboard. The ASCII code is returned in the lower 8 bits. The keyboard scan code is returned in the upper 8 bits.

USR3(1)---> Scans the keyboard buffer. Zero is returned if no key was struck. If there is a key in the buffer, the ASCII and scan codes are returned same as USR3(0), except the character will remain in the buffer.

USR3(2)---> Returns the current shift status. The bits returned are as follows:

*Bit7	=	Insert key active
*Bit6	=	Caps Lock key toggle
*Bit5	=	Num Lock key toggle
*Bit4	=	Scroll Lock key toggle
Bit3	=	Alternate key depressed
Bit2	=	Control key depressed
Bit1	=	Left Shift key depressed
Bit0	=	Right Shift key depressed

*Note: You can toggle the Insert, Caps Lock, Num Lock and Scroll Lock keys with the following statement:

```
POKE &17,  PEEK(&17,&40)  XOR  (2^bit), &40
```

(bit= 4-7 corresponding to the keys with bits 4-7 above).

EXAMPLE

```
DO
UNTIL USR3(1)<>0
A$=INKEY$
```

This is the same as `DO: A$=INKEY$: UNTIL LEN(A$)`, except the above example is much faster.

```
DO
   PRINT@(0,0) BIN$(USR3(2));
UNTIL INKEY$="Q"
```

This example will print the individual status bits on the screen. Pressing the keys specified in the list above shows the response of the status bits.

REMARK Remember that USR3(1) does not take the character out of the buffer. This can be useful for checking the keyboard for a specific key before going into a standard input routine, such as INPUT or INKEY$.

USR4 statement (Jump on CTRL C or CTRL BREAK)

FORMAT **USR4** (*address*)

DEFINITION USR4 is a predefined user function available on the IBM PC version. USR4 is used to set the ctrl-break address (or ctrl-C) when one is detected. The address specified must be in ZBasic's code segment.

EXAMPLE

```
USR4(LINE 20000)
```

This sets the ctrl-break address to be ZBasic's line 20000. In this case, if during program execution a ctrl-break is detected, a jump will be made to line 20000.

REMARK When USR4 is used to set the ctrl-break address, it must be understood that the program still cannot continue normal execution after the break is detected. The register and stack will be unpredictable; thus, the subroutine at the break address should finish with a stop or end to exit the ZBasic program.

USR5 function (Get Communication port status)

FORMAT USR5(-1 or -2)

DEFINITION USR5 is a *predefined* user function available on the IBM PC version. USR5 is used to return the status on either of the communication ports (see OPEN "C").

The status bits returned are defined as follows:

Line Status		
Bit 15	= Time Out	
Bit 14	= Trans Shift Register Empty	
Bit 13	= Tran Holding Register Empty	
Bit 12	= Break Detect	<---
Bit 11	= Framing Error	<--- These bits checked with ON COM
Bit 10	= Parity Error	<--- ERROR when a communication
Bit 9	= Overrun Error	<--- error occurs.
Bit 8	= Data Ready	

Modem Status	
Bit 7	= Received Line Signal Detect
Bit 6	= Ring Indicator
Bit 5	= Data Set Ready
Bit 4	= Clear To Send
Bit 3	= Delta Receive Line Signal Detect
Bit 2	= Trailing Edge Ring Detector
Bit 1	= Delta Data Set Ready
Bit 0	= Delta Clear To Send

EXAMPLE

```
J=USR5(-1)
```

This gets the status of the communications port 1 (COM1).

REMARK This function can be useful in investigating the RS-232 control. If there is a problem with the 232 communication (such as mismatched baud rate, parity error, time out, or cable hookup), it can be evident by observing the status via the USR5 function.

See "RS-232 COMMUNICATION" in this appendix for more information.

Also see OPEN"C", COM ON, COM OFF, COM START, COM END and ON COM ERROR GOSUB in this appendix. Also see OPEN"C" in the main reference section.

VARPTR function
VARSEG function

FORMAT **VARPTR** (variable)
VARSEG

DEFINITION This pair of functions is used to determine the memory address of a variable. VARPTR (*variable*) returns the offset of the variable. VARSEG returns the segment of*variable*.

EXAMPLE

```
DIM  A$(400)
A$ (400) = "HELLO"
A = VARPTR (A$(400)):      REM gets the offset
B = VARSEG:                REM gets the segment
PRINT PEEK(A,B):           REM print the length
PRINT CHR$(PEEK(A+1,B))    REM print the first char.

RUN

6
H
```

REMARK Because of the enhanced variable memory capability of this version of ZBasic, the VARPTR function is slightly different from previous versions. In earlier versions, the variables were put in one 64K segment which began at the data segment (DS). The address was returned in one 16-bit number.

Now in 4.0 integers, BCD's, strings and BCD/string arrays are given there own dedicated block of memory so that more data can be stored. Therefore, two numbers are needed; one for the offset; one for the segment.

The VARSEG function returns the segment of the last *variable* used with VARPTR. Its value is valid only immediately after a VARPTR has been executed.

VIEW PRINT statement

FORMAT **VIEW PRINT** *topline* **TO** *bottomline*

DEFINITION Used to the set scrolling bounderies.

topline	The top line to be used for output.
bottomline	The last line to be used for output.

This statement is very similar to the ZBasic DEF PAGE statement which also allows setting the column bounderies as well.

EXAMPLE

```
CLS
VIEWPRINT 5 TO 10
LOCATE 0,5  <--- Set cursor to first line position
DO
  PRINT"HELLO";
UNTIL LEN(INKEY$)
END
```

Demonstrates how the text is retained within the row limits.

REMARK Also see DEF PAGE.

WAIT statement

FORMAT **WAIT** *portnumber, AND expression* [, *XOR expression*]

DEFINITION Suspends program execution while checking the status of an input port.

portnumber	numeric from 0-65535.
AND expression	integer from 0-255
XOR expression	integer from 0-255

EXAMPLE

```
WAIT Port(x), 255,255
```

REMARK The WAIT statement causes execution of the program to be suspended until a specified port produces a certain bit pattern.

The data read at the port is XORed with the *XOR expression*, the ANDed with the *AND expression*. If the result is zero the program loops back to read the port again.

CAUTION: The computer may lock up if the required bit pattern does not appear on the port.

MSDOS APPENDIX

ZBASIC™

FULL SCREEN EDITOR

ZBasic 4.0 includes a powerful, yet simple, built in full screen editor for editing program text. To toggle between the Full Screen Editor and the Standard Line Editor, use the <F10> key.

DIFFERENCE BETWEEN THE FULL SCREEN AND STANDARD LINE EDITORS

STANDARD LINE EDITOR
This editor is provided for two reasons;

- The user may enter direct commands quite like a BASIC interpreter. You can enter things like PRINT ASC(A) and ZBasic will return 65. Math expressions may be entered like ?SQR(9) and 3 will be returned. See the section in the front of this manual call "Standard Line Editor" for detailed information about this editor.

- The Standard Line Editor works the same way on all versions of ZBasic inlcuding Apple //, CP/M, Z80 and Macintosh. This allows a common interface that someone may use that doesn't have the time to learn all the Full Screen Editors provided.

Note: If you opt to disable line numbers you will need to do all the editing in the Full Screen editor since EDIT requires a line number.

FULL SCREEN EDITOR
The full screen editor is provided to make entering and editing program code easy and fast. The following two pages describe the commands used with this editor.

continued...

MSDOS APPENDIX

FULL SCREEN EDITOR

Following is a list of command keys and control keys. Take a few minutes with the editor to become familiar with what these keys do.

MOVING THE CURSOR

KEY	CURSOR MOVEMENT
⇑	**UP** a line
⇓	**DOWN** a line
⇐	**LEFT** one character
⇒	**RIGHT** one character
Ctrl⇐	**LEFT A WORD**
Ctrl⇒	**RIGHT A WORD**
Home	**BEGINNING OF LINE**
End	**END OF LINE**
Ctrl-Home	**TOP OF TEXT** (beginning or start of program text)
Ctrl-End	**BOTTOM OF TEXT** (end of program text)
Pg Up	**PAGE UP**
Pg Dn	**PAGE DOWN**

DELETING TEXT

KEY	ACTION
Delete	**DELETE** character under cursor
Ctrl-E	**ERASE TO END OF LINE**

INSERT MODE or OVERWRITE MODE

KEY	ACTION
Insert	Toggles between **INSERT** mode and **OVERWRITE** mode. Insert mode inserts text at the current cursor position. Overwrite mode allows you type over the text under the cursor. (cursor is thicker in insert mode)

FULL SCREEN EDITOR COMMANDS

The Full Screen Editor commands are easy to learn. Spend a few minutes trying out the various commands.

You will notice that the function key commands are displayed on the bottom of the screen. When you press the ALT key the alternate set of commands is displayed.

FUNCTION KEY EDITOR COMMANDS

Editor Keys	Definitions
F1	LOAD file
ALT F1	SAVE file
F2	CUT line into buffer (for use with PASTE/ REPLACE)
ALT F2	PASTE line from buffer to the current line
F3	COPY line into buffer without erasing
ALT F3	REPLACE current line with line in buffer
F4	INSERT a new line after current line
ALT F4	DELETE current line
F5	FIND text. See FIND in main reference section
ALT F5	FIND NEXT occurence
F6	Set TAB value for indenting
ALT F6	AUTO TAB on/off. Toggles carriage return positioning
F7	RESTORE line to condition before changes made
ALT F7	LLIST to printer
F8	SCROLL screen text up (does not move cursor)
ALT F8	SCROLL screen text down (cursor doesn't move)
F9	FREEZE top of screen (press again to unfreeze)
ALT F9	FREEZE bottom of screen (press again to unfreeze)
F10	Return to Standard Line Editor
ALT F10	NEW. Erases all text in text buffer

Note: Programmers that use ZBasic on both the Apple and IBM should notice that the key sequences correspond to the open and closed Apple sequences so switching from one machine to another is much less confusing (of course the Standard Line Editor commands are the same).

Z80 APPENDIX

Z BASIC™

Z80™ Versions

TRS-80™ Models; 1, 3 and 4,
CP/M™-80 2.2, 3.0 and
CP/M™-80 Plus

by

Andrew Gariepy

ZEDCOR, INC.

Z80 APPENDIX

TABLE OF CONTENTS

Z80 APPENDIX

FILES ON THE MASTER DISKETTE

SYSTEM FILES

	FILENAME	DESCRIPTION
ZBASIC	ZBASIC.COM	This is the main ZBasic compiler and editor. Just type ZBASIC to execute. Note: On TRS-80 the file is called ZBASIC/CMD.

THE FILE ABOVE IS THE ONLY ONE REQUIRED TO RUN ZBASIC

THE FILES BELOW ARE OPTIONAL AND MAY BE DELETED FROM WORKING DISKETTES

EXAMPLE FILES

	FILENAME	DESCRIPTION
DEMO	ZBDEMO.COM	A limited demo version of ZBasic (public domain). Feel free to give it away to your friends, relatives and co-workers. This and ZBASIC.HLP may be given away together (please do not give away any other programs on this disk). Note: ZBDEMO13 on the TRS-80 model 4 version is the Model 1,3 demo.
HELP	ZBASIC.HLP	This is the help file. It is not necessary for using ZBasic but is helpful when learning ZBasic syntax. Just type HELP from the editor to get a menu of the help available.
DISK I/O	DISKIO.BAS	Example of using ZBasic disk file handling (DTEST on some disks)
GRAPHICS	ZROSE.BAS	Does a graphic "Rose" using High-speed and regular speed SIN and COSINE routines. (GTEST on some Z80 versions). Not included with all Z80 versions (storage limitations).
	CLOCK	Graphic clock example (most versions).
	APPLE.BAS	Bar and Line Graphs in Device independent Graphics.
SORT ROUTINES	SORT.BAS	This routine creates random data for arrays to demonstrate the SHELL and QUICK sort routines on this disk. Load this program first then do APPEND 1000 SHELL.APP (or QUICK.APP)
	SHELL.APP	The SHELL SORT that appears in the manual (under "Array variables.) A powerful sort when less items are used.
	QUICK.APP	The QUICK SORT that appears in the manual (under "Array variables.) A powerful sort when many elements need to be sorted.
MATH FUNCTIONS	SCIFN.APP	Examples of creating your own functions with ZBasic.

NOTE TO FILENAME SUFFIX SYNTAX: Filename suffixes will vary depending on the disk operating system being used. Syntax above is for CP/M. If you have a TRS-80 the suffix differs: ZBASIC.COM and ZBDEMO.COM become ZBASIC/CMD and ZBDEMO/CMD respectively. The period is changed to a slash for all the other files above; SORT.BAS becomes SORT/BAS...

Z80 APPENDIX

GETTING STARTED

TRS-80 MODEL 1

1. Make a backup and put the original in a safe place.
2. Read this appendix, making notes of any variations for the Model 1.
3. Follow the instructions under "Getting Started" in the main reference manual.

TRS-80 Model 3

1. Since this diskette is in TRS-80 Model one format, you MUST use the CONVERT utility to transfer the files on the master diskette to model 3 diskettes.
2. After converting the files make a backup of that diskette and put it and the master disk in a safe place for future use.
3. Read this appendix, making notes of any variations for the Model 3.
4. Follow the instructions under "Getting Started" in the main reference manual.

TRS-80 MODEL 4

1. Make a backup and put the original in a safe place.
2. Read this appendix, making notes of any variations for the Model 4.
3. Follow the instruction under "Getting Started" in the main reference manual.

CP/M-80 GENERIC

1. The diskette is provided on a Kaypro format Single sided-double density diskette. You will need a program like "Uniform™" to transfer the files over to your computer's diskette format if your drives will not read this diskette. ZBasic is also available on 8" format for an extra charge. Use the PIP utility to transfer files to your CP/M diskette.
2. Make a backup of your newly created master diskette and put it and the original in a safe place for future use.
3. See "Special Configuration Options" in this appendix to set up ZBasic to work with your terminal type. You will probably need your terminal or computer hardware reference manual for control codes for Clear screen, cursor control and the sort.
4. Read this appendix, making notes of any variations for the Model 3.
5. Follow the instruction under "Getting Started" in the main reference manual.

KAYPRO® CP/M

1. Make a backup of your master diskette and put it, and the original, in a safe place for future use.
2. Read this appendix, making notes of any variations for the Model 3.
3. Follow the instruction under "Getting Started" in the main reference manual.

Z80 APPENDIX

NOTES ON THE Z80 VERSION

MEMORY

ZBasic 3.0 for the Z80 Versions is designed to run in a DOS environment (CP/M-80 or TRSDOS). The typical programming area available in a 48k to 64k machine is from 24k to 56k. ZBasic also has the capability of OVERLAYS which are explained later in this Appendix (this version will only use a single bank of 64k.)

FILENAMES

The filenames in ZBasic are the standard DOS filenames. (Disk Operating System). Example:

TRSDOS™

```
ZBASIC/CMD        PROGRAM/BAS:1        MYPROG/BAS.SECRET:0
```

CP/M™-80

```
ZBASIC.COM        A:PROGRAM.BAS        B:MYPROG.BAS
```

SOUND

TRS™-80 Model 4, 4p
The range of frequencies for the internal speaker of the model 4 are limited to: **Ø,Ø** through **7,31**.

TRS™-80 Model 1&3
The frequency range is from 100hz to 10,000 hz and is routed out the out the cassette port. Connect a speaker amplifier to the cassette port to get sound. See your Radio Shack dealer for pricing (about $10).

CP/M™-80 & Kaypro™
Since most CP/M systems do not support sound, SOUND is routed as CHR$(7) (tone will not vary). Check your users manual for sound capabilities and porting. Sometimes OUT (n) may be used.

Z80 ICON

Whenever you see this icon in the main reference section of the manual take note of it's contents. It is referring to a variation in the use of that command for one of the Z80 versions.

ZBASIC DEMO PROGRAM

There is a ZBasic demo program on your disk that may be copied and given away to friends called ZDEMO.COM or ZDEMO/CMD. This is a limited version of ZBasic that contains all the functions and is only limited by program size and saving object code. Feel free to give copies of the ZDEMO program and the ZBasic.HLP program to your friends.

continued...

Z80 APPENDIX

Notes on Z80 continued

SCHOOLS

Schools may duplicate the ZDEMO/CMD program for teaching. This program is very useful in a classroom since, in most cases, a full blown language is rarely necessary. Note that this also cuts down the costs to schools (under no circumstances may the actual ZBasic program be copied for distribution).

GRAPHICS Special graphics Modes:

TRS-80™ Model 4, 4p
This version will use the Radio Shack™ Model 4, 640 x 240 high-resolution graphics board and the Micro-Labs™ high-resolution board in MODES 8 through 15 only.

TRS-80™ Model 1& 3
No High resolution modes are supported.

CP/M™-80
Graphics are not supported. All Graphics are emulated using text characters. See COLOR and CLS for changing the character type being used.

KAYPRO II, 4, 10 Special Graphics version
Kaypro 160x100 Graphics are supported with this special CP/M version. Your Kaypro must have graphics for this version to work.

MOUSE Does not function with this version. See "Patch" in this appendix for ways a mouse can be patched in.

HELP FILE The file used in the ZBasic HELP command is called ZBASIC/HLP or ZBASIC.HLP. This file may be deleted to allow more room on the disk. If HELP is not on the disk, typing HELP from the editor will generate a "FILE NOT FOUND" error.

CALL *"string"* Supported only in the TRS™-80 Versions. This passes a DOS command to the operating system. Example: CALL "DIR".

COLOR *number* All present Z80 versions use Ø (zero) as BLACK. Any other value will be WHITE or the "*" character.

TIME$/DATE$ These commands are supported on the TRS-80 only. CP/M versions will return ØØ:ØØ:ØØ and ØØ/ØØ/ØØ respectively.

Z80 APPENDIX

Z80 DISK FORMATS

The Z80 versions of ZBasic are provided on a specific disk format depending on the machine. The format descriptions are as follows:

TRS-80 MODEL 1 & 3

Format: TRS™-80 Model 1
5 1/4 inch, 35 track, Single Sided, Single Density

Model 1
Transfer: Model 1 TRSDOS 2.3
Boot ZBasic™ disk in drive 0. The disk will copy all files to a formatted TRSDOS™ 2.3 System Disk

Model 3
Transfer: Model 3 (4 in 3 mode) TRSDOS 1.3
Use TRSDOS™ 1.3 or compatible CONVERT command to move files from the model 1 formated diskette to a model 3 compatible format. See instructions for CONVERT in your TRSDOS manual.

TRS-80 MODEL 4 & 4p

Format: TRS™-80 Model 4 & 4p TRSDOS 6.0.2
5 1/4 inch, 40 track, Single Sided, Double Density (TRSDOS™ 6 Format)

Transfer: Just copy files to your system disk.

OTHER TRS-80 OPERATING SYSTEMS

ZBasic can be copied over to most TRS-80 Disk Operating Systems like MultiDos, DosPlus, Newdos, and LDOS.

CP/M-80 VERSIONS 2.2, 3.0 or PLUS

Format: CP/M™-80 version 2.2 or 3.0
5 1/4 inch, 40 track, Single Sided, Double Density (Kaypro™ II Format)

Transfer: If your computer cannot copy the ZBasic files over to your format using PIP, try using a transfer program to move files onto your disk format. Some popular transfer programs: Interchange™, Multidisk™, Uniform™.

Once ZBasic is in your disk format it may copied like other files.

SPECIAL KAYPRO GRAPHICS VERSION

Format: KAYPRO™, CP/M™-80 version 2.2
5 1/4 inch, 40 track, Single sided, Double Density

Transfer: Use PIP to transfer programs onto your system disk.

Z80 APPENDIX

Z80 OPTIONAL BINARY MATH PACKAGE

Zedcor offers an optional Binary Math package that allows you to get faster execution times when doing floating point math. Contact Zedcor at 1-800-482-4567 if you want this optional package.

While high speed floating math may be desirable, there are a number of trade-offs:

- Speed increase of 10x is typical.
- Precision is not definable like BCD versions of ZBasic. Digits of accuracy for both single and double precision is 6.2 digits with a range of E±38.
- Binary constants and variables require four bytes each for RAM and DISK storage.
- Binary numbers are stored in a different format that BCD numbers.
- MKB and CVB work with binary numbers only.
- Speed increases are up to 10 times the speed of the BCD version. There is the typical binary rounding error factor (not in the regular version of ZBasic).

Note: Programs created with the binary math version of ZBasic cannot read files with BCD floating point created with the BCD version and vice-versa. The binary format is subject to change in future and other versions of ZBasic. Do not use PEEK or POKE on binary variables.

Note: A# and A! are different variables (even though single and double are the same precision).

Z80 APPENDIX

EXECUTING ZBasic™

There are basically two ways of starting ZBasic from the operating system prompt. With CP/M the DOS prompt is A>or C> depending on the drive used. With TRSDOS it is DOS Ready.

1. **Type:** ZBASIC <ENTER>

   This is the standard way to startup ZBasic. See "GETTING STARTED" in the ZBasic standard manual. Also see "CONFIGURATION" in this appendix.

2. **Type:** ZWARM <ENTER>

   This is a special way to startup ZBasic to recover a ZBasic text program after a crash or reset . A version of ZBasic must be created using the <W>arm start creator option from the ZBasic start up screen.

   To Create this WARM start version configure ZBasic for your machine and save using the <S>ave option from the ZBasic startup menu. (DO NOT use your MASTER DISK only use a BACKUP COPY of your master diskette).

   Exit ZBasic using QUIT and re-enter the just created configured ZBasic and use the <W>arm Start Creator option to create a WARM start version of ZBasic to be called (TRS™-80: **ZWARM/CMD** , CP/M™-80: **ZWARM.COM**) which can be used to recover ZBasic program text after a RESET or program lock-up.

Important Note: ZWARM will only recover a program if it is still intact in memory and has not been overwritten. This will not recover from a **NEW** as it erases the program.

The ZBasic 3.0 patch option allows the user to PATCH specific addresses in ZBasic to Change areas in the JUMP TABLE for special hardware or software and to apply fixes to the actual program as specified by Zedcor to provide some special features. These changes may be saved by using the <S>ave option from the start-up MENU.

Z80 APPENDIX

Z80 SPECIFIC CONFIGURATION OPTIONS

After typing "C" in the initial prompt screen, ZBasic will ask for the standard configuration parameters explained in the "Getting Started" section of the manual. Following these standard parameters are the Z80 specific configuration parameters. The additional prompts displayed are as follows:

NOTE: Press <ENTER> to skip options. Press <BREAK> to exit (<CTRL C> with CP/M).

CONFIGURE OPTIONS ON ALL Z80 VERSIONS

```
Default  Clear  nnnnn  Size              03E8    01000   ?
```

This selects the default amount of memory to be set aside for strings in the INDEX$ area at compile time. The actual amount of memory in the running compiled program can be found by using MEM function in the program. If this area becomes less than or equal to zero due to high memory drivers a 'Not Enough Memory' error will be displayed and the program will exit back to DOS.

```
LIST  First  Line              <Key>   0019    00025
```

When pressed as the first key on a line will cause the editor to LIST the first program line and make it the current line. Typical key: <Left Arrow>

```
LIST  Last  Line              <Key>   012C    00300
```

When pressed as the first key on a line will cause the editor to LIST the last program line and make it the current line. Typical key: <Right Arrow >

```
LIST  Previous  Line          <Key>   012C    00300
```

When pressed as the first key on a line will cause the editor to LIST the previous program line and make it the current line. Typical key: <Up Arrow >

```
LIST  Next  Line              <Key>   012C    00300
```

When pressed as the first key on a line will cause the editor to LIST the next program line and make it the current line. Typical key: <Down Arrow>

```
FIND  Next  occur.            <Key>   003B    00059
```

When pressed as the first key on a line will cause the editor to FIND the next occurrence of the string last used with the FIND command and make it the current line. Typical key: <semi-colon>

```
Overlay Offset (see "Overlays"):      00000   00000
```

Allows you to set the Offset for overlay programs. See OVERLAYS in this appendix for details.

continued...

SPECIAL TRS-80 CONFIGURATION OPTIONS

`TRSDOS, NEWDOS or Other <T/N/O>  012C 00300`

This special Configuration is used to tell the EDITOR which type of DOS you are using so the **DIR** command will be available from the editor.

Type a **"T"** if you are using TRSDOS.

Type an **"N"** if you are using NEWDOS (be sure to re-enable the BREAK key with NEWDOS; see your NEWDOS manual for details).

Type an **"O"** for using most OTHER TRS-80 type disk operating systems.

If not configured correctly, a system crash may occur when **DIR** is used from the editor. This is one of those things in machine language that was never truly standardized by TRSDOS and other Disk Operating Systems.

DIR does not function from the editor with Model 1 TRSDOS or NEWDOS. Most other Model 1 Disk Operating Systems like LDOS, MultiDOS etc. function properly.

DOS COMMANDS FROM THE TRS-80 VERSIONS ONLY

To use DOS commands from within your programs use **CALL**"DOS Command". To do a DIR from within a ZBasic program use **CALL"DIR"**. To find out how much disk space is available use **CALL"FREE"**, etc.

Note: The DOS function being called *MUST NOT* use memory over 5200H for Model 1 or 3 and 3000H for Model 4. This may not work with some disk operating systems.

Note: NEWDOS stops the system scan of the <BREAK> key. Use the NEWDOS: **SYSTEM BREAK ON** command (or whatever command that particular version of NEWDOS uses. See your NEWDOS MANUAL).

continued...

Z80 APPENDIX

SPECIAL CP/M CONFIGURATION OPTIONS

```
PRINT@ Control (Hex Code)          3D1B       15643
```

This Configuration question tells ZBasic which control codes for the screen will cause the cursor to be positioned for used with the **PRINT@** or **LOCATE** function. The codes for this can be found in your computer terminal technical manual. If a single character just type the character code in decimal or Hex (precede the Hex code with a "&").

If the code is two characters like 1B and 54, type the number in Hex in reverse order; &541B. These codes must be correct for the ZBasic text graphics or screen **PRINT@** or **LOCATE** functions to operate.

```
Clear Screen String (Hex Code)     001A     00026
```

This Configuration question tells ZBasic which control code for the screen will cause the screen or terminal to be cleared of text and graphics using **CLS**. The correct codes for this can be found in your computer or terminal technical manual. If a single character just type the character code in decimal or Hex (precede a Hex code with a "&").

If the code is two characters like 1B and 54, type the number in Hex in reverse order; &541B. These codes must be correct for the ZBasic text graphics or screen **CLS** function to operate.

```
Clear To End of Line (Hex Code)    541B     21531
```

This Configuration question tells ZBasic which control code for the screen will clear the text and graphics from the cursor position to the end of the line using **CLS LINE**. The correct codes for this can be found in your computer or terminal technical manual. If a single character just type the character code in decimal or Hex (precede a Hex number with a "&").

If the code is two characters like 1B and 54, type the number in Hex in reverse order; &541B. These codes must be correct for the ZBasic text graphics or screen **CLSLINE** function to operate.

```
Clear End of page (Hex Code)       591B     22811
```

This Configuration question tells ZBasic which control code for the screen will clear the screen from the cursor position to the end of the screen using **CLS PAGE**. The correct codes for this can be found in your computer or terminal technical manual. If a single character just type the character code in decimal or Hex (precede a Hex number with a "&").

If the code is two characters like 1B and 54, type the number in Hex in reverse order; &541B. These codes must be correct for the ZBasic text graphics or screen **CLSPAGE** function to operate.

Note: Also see JUMP TABLE and PATCH in this appendix for configuring control strings longer then two characters.

continued....

Special CP/M™-80 Configurations continued

`Cursor (off) String (Hex Code)  451B 17691`

This Configuration question tells ZBasic which control code for the screen will turn on the blinking cursor using **LOCATE x,y, OFF**. The correct codes for this can be found in your computer or terminal technical manual. If a single character just type the character code in decimal or Hex (precede a Hex code with a "&").

If the code is two characters like 1B and 54, type the number in Hex in reverse order; &541B. These codes must be correct for the ZBasic text graphics or screen **LOCATE x,y, OFF** function to operate.

`Cursor (on) String (Hex Code)  521B  21019`

This Configuration question tells ZBasic which control code for the screen will turn on the blinking cursor using **LOCATE x,y, ON**. The correct codes for this can be found in your computer or terminal technical manual. If a single character, just type the character code in decimal or Hex (precede a Hex code with a "&").

If the code is two characters like 1B and 54, type the number in Hex in reverse order; &541B. These codes must be correct for the ZBasic text graphics or screen **LOCATE x,y, ON** function to operate.

NOTE: IF these parameters are not set properly the corresponding functions will not operate.

Z80 APPENDIX

USING THE PATCH UTILITY

The Z80 versions of ZBasic™ provide a utility to Patch or modify ZBasic internal code to allow fixes or modifications for specific Hardware or software.

To get into the PATCH mode enter ZBasic™ from DOS and use the <P>atch menu option. You will then be prompted for an address which may be decimal (or HEX if preceded by a '&' character). The Modifications made during the patch session may be saved by using the <S>ave option when completed. Some example patches are shown below.

<P>atch	Enter PATCH mode from ZBasic start-up.
Address:	Enter Address to View and/or Patch Data <BREAK> to Abort to menu
aaaa_bb ?_ (aaaa=hex address) (bb= byte at address)	Enter data to change <ENTER> to Skip <BREAK> to Abort and go back to Address: If Data or <ENTER> selected the next address will be shown.

BOLDFACE text is what you type in.

• COORDINATE WINDOW Patch (enable pixel graphics)

<P>atch	get into patch mode
Address: **&xx3F**	Jump table address for x,y conversion
xx3f_C3?_ **&C9**	Change JP to RET
Address: **<BREAK>**	
<BREAK>	
<E>dit	

• Patch to route MOUSE(x) to a user routine.

<P>atch	get into patch mode
Address: **&xx3D**	Jump table address for JP mouse
xx3f_C3?_ **&C9**	Change JP to RET
Address: **<BREAK>**	
<BREAK>	
<E>dit	

• Patch to Set the default USR3 vector.

<P>atch	get into patch mode
Address: **&xx61**	Jump table address for USR3(expr)
xx3F_C3?_ **&ØØ**	Change JP to address FØØØ
XX40_FF?_ **&FØ**	
Address: **<BREAK>**	
<BREAK>	
<E>dit	

xx=01 for CP/M™-80
xx=30 for TRS-80™ Model 4 TRSDOS/LDOS 6.2
xx=52 for TRS-80™ Model 1 and 3

Z80 APPENDIX

MEMORY CONSIDERATIONS

The Z80 versions of ZBasic have three different modes of operation concerning memory organization -- EDIT mode, RUN mode, and RUN* mode (see memory map on following page).

At least 32k of free memory is required for the EDIT and RUN modes (the development stage of the program). However, after a program has been compiled and saved using RUN*, it can be run on as little as 16k of free memory depending on the size of the program (the RUN* mode shown on the memory map).

NOTES ON THE Z80 MEMORY MAP:

- The system top of memory is observed by ZBasic in both the editor and object code. The CLEAR area in the /CMD or .COM file created by ZBasic is the only area of the compiled program which can adjust to different sizes of high memory drivers or machine language routines. If this area is too small when an attempt is made to execute this program from DOS a "Not Enough Memory" error will occur and it will return to DOS.

- The ZBasic subroutines and jump tables are not saved to disk when a program is compiled as a chain file using RUN+. Thus, chain files take up 10k less on disk.

- ZBasic is located immediately after the DOS. There may be drivers or other applications installed at the top of memory. ZBasic does, however, assume to own all of the memory from DOS to the TOP of memory as defined by the DOS.

- The size of the INDEX$ is determined by the **CLEAR** statement (see reference section). This version defaults to **CLEAR 1000**, making the INDEX$ equal to 1k. If there is not enough memory , the largest size available will be allocated. The size of the INDEX$ memory can be determined using the MEM function within the running program.

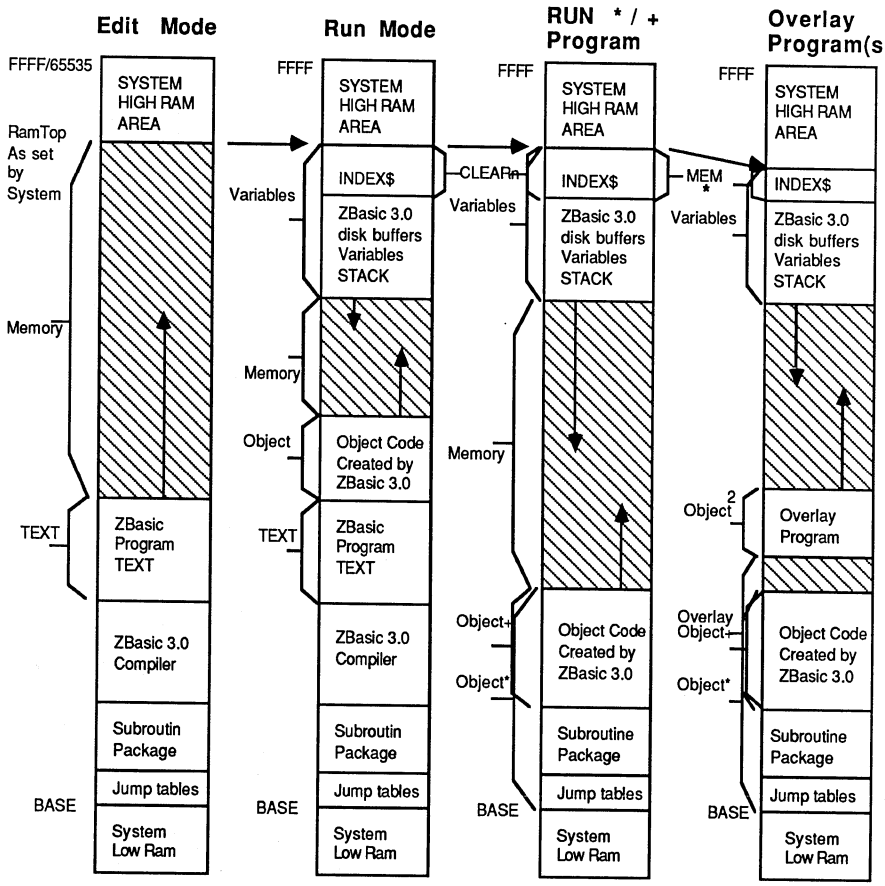
- When the **CALL *string*** statement is used to execute a DOS function the DOS function must not use the area where ZBasic resides otherwise a system crash may occur (this DOS function jump vector is located in the ZBasic jump table so it may be re-vectored for different Disk Operating Systems).

- TopRam is the highest RAM address the system will allow ZBasic to use. This address varies from system to system and even on the same system depending on the DOS.

  The INDEX$ CLEAR area is the only area of ZBasic that can adjust to this area if not enough room is allocated. When the object code is executed, a "Not Enough Memory" error will occur and the object code will stop and return to the operating system. MEM will return memory for INDEX$.

Z80 APPENDIX

Z80 MEMORY MAP



BASE ADDRESSES

CPM80 2.2 or 3.0	0100 HEX OR 256	DECIMAL
TRS80 Model 1 or 3	5200 HEX OR 21976	DECIMAL
TRS80 Model 4 ver.6.2	3000 HEX OR 12288	DECIMAL

Memory available for ZBasic Text and ZBasic Object Code.

Z80 APPENDIX

RS-232 COMMUNICATION

TRS-80 Model 1, 3 and 4 SERIAL INTERFACES

ZBasic for the TRS-80 Model I, III and 4 supports asynchronous communication using the filenumber **-1** (negative one) for the standard serial Interface.

Baud rate, parity, stop bits, and word length are all controlled in the OPEN "C" statement (see OPEN "C" in the reference section).

CP/M™-80 SERIAL INTERFACES

The serial interface on Kaypro™ and CP/M™-80 attempts to use the **CP/M TTY device** and the OPEN"C" does not affect parameters as these are not software selectable.

See your CP/M terchnical reference manual for changing parameters of the TTY device. Use the Patch option to add your own Machine language drivers to the jump table.

SERIAL COMMUNICATION PROBLEMS

If asynchronous communication is not working, try one of the following:

• Check to make sure the baud rate, parity, stop bits, and word length settings are the same on both sides of the communication.

• Check for proper cable wiring. The cable must support the standard RS-232 asynchronous interface. If the serial transfer works at a low baud rate (like 300 baud) but fails at higher baud rates, the cable is probably wired improperly.

See the diagrams which shows the two most typical cable configurations in the MSDOS appendix.

Z80 APPENDIX

Z80 JUMP TABLE

The Z80 versions make available a jump table that can be altered to route routines to other addresses. This can be useful for implementing special software or for handling non-compatible Disk Operating Systems or Serial ports (etc.).

The following is a list of all the available jump locations with a short description of each:

Address	Vectors	Description	Special Notes
XX00	JP SUBRTN	Cold Start entry point	NONE
XX03	JP WARM	Warm Start entry point	NONE
XX06	JP EXIT	Exit to System	NONE
XX09	JP CHRINP	Get Keyboard Character	* RETURNS KEY IN A
XX0C	JP VIDOUT	Output Character to display	* SENDS CHAR IN A
XX0F	JP LPROUT	Output character to printer	* SENDS CHAR IN A
XX12	JP SCANKY	Get Key from kybrd no waiting	* A=0 NO KEY ELSE A=KEY
XX15	JP ICOMM	INITIALIZE RS232 PORT	Set Baud,Parity,Stop bits,Word len UAR
XX18	JP BAUD	SET RS232 BAUD RATE	HL=BAUD RATE 300, 1200..
XX1B	JP PARITY		SET RS232 PARITY, HL=PARITY 0, 1, 2
XX1E	JP STOPBT	SET RS232 STOP BITS	HL=STOP BITS 0, 1, 2...
XX21	JP WORDLN	SET RS232 WORD LENGTH	HL=WORD LEN 5, 6, 7, 8...
XX24	JP RSREAD	READ 1 CHAR FROM RS232	*On exit A=char from RS-232 set Z flag
XX27	JP RSWRT	WRITE 1 CHAR TO RS232	*A=CHAR TO SEND set Z flag on exit
XX2A	JP RSSCAN	GET RS232 CHR NO WAIT	* A=0 NO Char,Z flag set else A=CHAR
XX2D	JP TIME	GET TIME STRING	ON Exit HL >=> 9 bytes: lenHH:MM:SS
XX30	JP DATE	GET DATE STRING	ON Exit HL >=> 9 bytes: lenMM/DD/YY
XX33	JP PRINTA	PRINT @(X,Y) ROUTINE	HL=Y,DE=X Set cursor for next characte
XX36	JP PRINTG	PRINT %(X,Y) GRAPHIC	Loc HL=Y,DE=X set cursor for next char
XX39	JP SOUND	GENERATE SOUND	HL=DURATION MS,DE=FREQ in HZ
XX3C	JP MOUSE	READ MOUSE STATUS	HL=TYPE RETURNS IN HL
XX3F	JP CONVXY	CONVERT GRAPHIC POS	HL=Y,DE=X RETURN IN HL, DE
XX42	JP MODE	SET GRAPHICS MODE	HL=MODE #
XX45	JP COLOR	SET COLOR FOR GRAPHICS	HL=COLOR
XX48	JP SETXY	SET POINT DE,HL	LOCAL COORDINATES
XX4B	JP PLOTXY	PLOT FROM LAST POINT TO	DE, HL LOCAL
XX4E	JP FILLXY	FILL FROM POINT	DE, HL FILL AREA around X,Y
XX51	JP POINT	READ COLOR AT POINT	DE,HL RETURNS HL=COLOR
XX54	JP DOSCALL	CALL DOS COMMAND	HL >==> STRING

*save registers

XX=01 for CP/M80 2.2 & 3.0
XX=30 for TRS-80 Model 4 and 4P, TRSDOS/LDOS 6.2
XX=52 for TRS-80 Model 1 and 3 versions

DEFAULT USR TABLE

USR *digit (expression)*

Entry: expression >==> Z80 register HL
Exit: HL register returned in the *expression* contained in the USR function.

Address	Vectors	Description	Special Notes
XX57	JP **USR0**	OLD HL RETURNS	HL ON COLD START ENTRY
XX5A	JP **USR1**	RETURN	
XX5D	JP **USR2**	RETURN	
XX60	JP **USR3**	RETURN	
XX63	JP **USR4**	RETURN	
XX66	JP **USR5**	RETURN	
XX69	JP **USR6**	RETURNS LAST LINE # EXECUTED WITH TRON ACTIVE	
XX6C	JP **USR7**	RETURNS RAW RANDOM # 0 TO 65535	
XX6F	JP **USR8**	RETURNS SIN OF HL IN BRADS AS A VALUE OF +/- 256	
XX72	JP **USR9**	RETURNS COS OF HL IN BRADS AS A VALUE OF +/- 256	

SPECIAL STRINGS AND CONSTANTS

These Strings and constants may be changed by POKES or using the PATCH function from the MENU.

XX80	CLSSTR	4 BYTE CLEAR SCREEN STRING
XX84	PRNTAM	6 BYTE PRINT AT CONTROL STRING <ESC>, <=>, Y+32, X+32, 0, 0 (TRS-80 versions first 2 bytes are RS232 configuration bytes)
XX8A	CUROFF	4 BYTE CURSOR OFF STRING
XX8E	CURON	4 BYTE CURSOR ON STRING
XX92	CLRLINE	4 BYTE CLEAR TO END OF LINE STRING
XX96	CLRPAGE	4 BYTE CLEAR TO END OF PAGE STRING
XX9A	PAGE0	1 BYTE PRINTED LINES PER PAGE (0=disabled)
XX9B	PAGE1	1 BYTE TOP MARGIN (0=none)
XX9C	PAGE2	1 BYTE ACTUAL PAGE LENGTH IN LINES (0=disabled)
XX9D	PAGE3	1 BYTE PRESENT LINE (line#1=0, line#2=1....)

OTHER IMPORTANT ADDRESSES

52A0 and 52A2	------> TRS80 Model 1, 3, 4 High resolution 240,640 in words
XXA4 TO XXBF	------> User area for PATCHES (Saved with <S>ave configuration option)
XX00 +200H	------> 256 buffer (OK to use whole buffer during machine language routine)

XX=01 CP/M80 2.2 & 3.0
XX=30 TRS-80 Model 4 and 4P, TRSDOS/LDOS 6.2
XX=52 TRS-80 Model 1 and 3 versions

Z80 APPENDIX

USING OVERLAYS

ZBasic 3.0 on Z80 based computers allows for Overlays to be used to make the most efficient use of a systems available memory.

An Overlay is a program which is loaded from disk (without destroying the program in memory) and executed. After it is executed, it will RETURN to the main program. As long as it is in memory it may be called over and over again by RUN 0(zero) until it is overwritten by another overlay or program.

The main advantage of Overlays is they are small and will normally load up quickly. After they have been loaded, they work like a GOSUB with the variables being chained that appear in the DIM statements at the start of the main program and the overlay program.

Here are the steps in creating an overlay program.

1. Create the MAIN program and define the Overlay subroutine(s).

2. To determine the *OFFSET* for the overlay subroutine:

   A. Type **RUN+** from the MAIN program
   (type **<ENTER>** when it asks for a filename)

   B. Type: **MEM**

   ```
   00000 Text
   00000 Memory    <--- This is the room available for the overlay.
   00000 Object    <--- This, plus 100, is the OFFSET amount.
   00000 Variable
   ```

   Add 100 to the number in front of Object to get the offset. This what is used in the <C>onfigure startup to create the OFFSET for the overlay program.

3. Set up all variables which will be used by the Overlay program in identical DIM statements at the start of both the MAIN program and OVERLAY subroutines.

4. Compile and save the **MAIN** program using the **RUN*** command.

5. Compile and save the **OVERLAY** subroutine using **RUN+.**

6. When the MAIN program requires the use of the overlay the first time use:

   OPEN"I",1,"Overlay Filename": RUN 1

7. After the overlay is loaded it may be executed again *without* reloading the OVERLAY by using the RUN Ø (zero) statement.

   See the OVERLAY program examples on the next page...

OVERLAY PROGRAM EXAMPLE

To see how the overlay capabilities work try typing in these program examples as shown. <cr> is the <ENTER> key.

MAIN Program

```
ZBASIC<cr>
(in configure)
OVERLAY OFFSET = 0
E
ZBasic Ready

00010 CLEAR 5000 : DIM E,X,Y,Z,T$
00020 PRINT "STARTING MAIN PGM"
00030 OPEN"I",1,"PGMOVL"
00040 PRINT "GOING TO OVERLAY"
00050 E=0 : RUN 1
00060 PRINT "BACK FROM OVERLAY"
00070 E=1 : RUN 0
00080 E=2 : Y=1 : Z=8 : RUN 0
00090 E=3 : RUN 0
00100 PRINT "T$=";T$;"'"
00110 STOP



RUN+
<cr>  don't Save Object!!!
MEM
00217 Text
----- Memory
00200 Object
06000 Variable

RUN*      (TRSDOS)  (CPM80)
Object File.. MAIN/CMD or MAIN.COM

MEM
00217 Text
----- Memory
11000 Object  >==> (size on disk)
06000 Variable

QUIT<cr>


(Compile overlay program)
```

OVERLAY Program

```
ZBASIC<cr>
(in configure All Else Same!)
OVERLAY OFFSET = 200+256 (aprox)=500
E
ZBasic Ready

00010 CLEAR 5000 : DIM E,X,Y,Z,T$
00020 IF E=0 THEN PRINT "OVERLAY"
00030 ON E GOTO "HELLO","TEST",120
00040 PRINT ""BAD COMMAND""
00050 RETURN
00060 "HELLO"
00070 PRINT "HELLO" : RETURN
00080 "TEST"
00090 FOR X=Y TO Z
00100     PRINT X,
00110 NEXT X : PRINT : RETURN
00120 T$=STRING$(20,"X")
00130 RETURN

RUN+
Object Filespec.... PGMOVL<CR>

MEM
00208 Text
----- MEMORY
00150 Object >==> (size on disk)
06000 Variable

QUIT<cr>

MAIN
STARTING MAIN PGM
GOING TO OVERLAY
IN OVERLAY
*BAD COMMAND*
BACK FROM OVERLAY
HELLO
1  2  3  4  5  6  7  8
T$='XXXXXXXXXXXXXXXXXXXX'
Break in 00110
```

Z80 APPENDIX

Z80 MODE CHART

ZBasic™ Graphics Mode Chart

Mode	Graphics Mode		CPM™-80 Z80 ver. 2.2,3.0		Kaypro™ With Graphics		TRS™-80 Model 1&3		TRS™-80 Model 4 & 4p	
			Print	Plot	Print	Plot	Print	Plot	Print	Plot
0	Text	x	80	80	80	40	32	32	40	40
		y	24	24	24	12	16	16	24	24
1	Graphics	x	80	80	80	80	32	64	40	80
		y	24	24	24	50	16	48	24	72
2	Text	x	80	80	80	60	32	32	40	40
		y	24	24	24	18	16	16	24	24
3	Graphics	x	80	80	80	120	32	64	40	80
		y	24	24	24	72	16	48	24	72
4	Text	x	80	80	80	80	64	64	80	80
		y	24	24	24	24	16	16	24	24
5	Graphics	x	80	80	80	160	64	128	80	160
		y	24	24	24	96	16	48	24	72
6	Text	x	80	80	80	80	64	64	80	80
		y	24	24	24	25	16	16	24	24
7	Graphics	x	80	80	80	160	64	128	80	160
		y	24	24	24	100	16	48	24	72
8	Text	x	80	80	80	40	32	32	40	40
		y	24	24	24	12	16	16	24	24
9	Graphics	x	80	80	80	80	32	256*	40	320*
		y	24	24	24	50	16	192*	24	240*
10	Text	x	80	80	80	60	32	32	40	40
		y	24	24	24	18	16	16	24	24
11	Graphics	x	80	80	80	120	32	256*	40	320*
		y	24	24	24	72	16	192*	24	240*
12	Text	x	80	80	80	80	64	64	80	80
		y	24	24	24	24	16	16	24	24
13	Graphics	x	80	80	80	160	64	512*	80	640*
		y	24	24	24	96	16	192*	24	240*
14	Text	x	80	80	80	80	64	64	80	80
		y	24	24	24	25	16	16	24	24
15	Graphics	x	80	80	80	160	64	512*	80	640*
		y	24	24	24	100	16	192*	24	240*

* Model 4 Radio Shack™ or Micro Labs™ Graphics Board Only. (Do not use Modes 8 to 11)
Note: ZBasic™ will not scale High resolution graphics correctly on the Model 4 in 3 mode.

x=Horizontal resolution, y=vertical resolution
NOTE: ZBasic does not support both text and graphics in MODE 13 or 15 (Micro-Lab's board does).

APPLE DOS 3.3 APPENDIX

Notes

APPLE DOS 3.3 APPENDIX

DOS 3.3 Version

For the Apple® *IIe, IIc, Laser 128™* and IIGS

(//e requires the extended 80 column card)

Original version
by

David Overton

Enhancements
by
Greg Branche

ZEDCOR, INC.

APPLE DOS 3.3 APPENDIX

TABLE OF CONTENTS

APPLE DOS 3.3 APPENDIX

HARDWARE REQUIREMENTS

APPLE *//c*, IIGS, LASER 128™

The Apple™ version of ZBasic™ 3.2 functions with a standard Apple *//c* or IIGS. A 5.25" disk drive is required.

An Apple™ Mouse, second Disk drive and Joystick are supported but are not required.

The IIGS emulates the //e and //c with this version. IIGS specific features are not used.

APPLE *//e*

The Apple™ version of ZBasic™ 3.2 for the Apple IIe requires an ***Extended 80-column card installed in auxiliary slot 3 and 1 disk drive.***

An Apple™ Mouse w/interface, Joystick and Super Serial Card are supported but are not required.

CREATING PROGRAMS TO EXECUTE ON AN APPLE][+ RUNNING DOS 3.3

ZBasic requires a minimum of 128K memory to create programs, but compiled programs will normally function on the older machines with 64K since the Object Code (Machine language program) is compiled into the lower bank of 64K.

Code can be generated which will run on an Apple™ II+ if certain restrictions are observed. Avoid the use of MODE 3 or 7 as they require an extended 80 column card which will not function in an Apple II+.

The Apple II+ MUST have a 16k memory card installed in slot # zero in order to execute programs created with ZBasic (total of 64K memory).

VIDEX™ 80 COLUMN CARD

The Videx 80 column card works in MODE 2. You may need to clear the screen with CHR$ (n). See 80 column manual for value of n (usually 12). Older style 80 column cards may or may not function.

ProDOS™

Another option is to create your programs with the 64K ProDOS version of ZBasic. Programs created with this version will run on any Apple //. You may order the ProDOS version from Zedcor at 800-482-4567.

APPLE DOS 3.3 APPENDIX

FILES INCLUDED ON MASTER DISKETTE

The following files are included on your master copy of ZBasic for Apple // DOS 3.3:

Files	Description
A 002 HELLO	Hello program starts up ZBasic when you boot the disk.
*B 037 ZBASIC	The ZBasic master program
*B 003 INTERPILER	Part of the main program
*B 042 VERSION 3.2	"" ""
*B 026 COPYRIGHT 1985	"" ""
*B 011 BY	"" ""
*B 006 ZEDCOR, INC.	"" ""
*B 047 ALL RIGHTS	"" ""
*B 010 RESERVED	"" ""
*B 013 APPLE //TM128K	"" ""

The files above are required to create ZBasic programs.

The files below are optional and are not required to execute ZBasic.

Example Files	Description
*T 133 ZBASIC.HLP	The ZBasic HELP file accessed from ZBasic with HELP. If file is not located you will get a File-Not-Found error.
T 003 GRAPH.BAS	Example of graphs.
T 013 DISKIO.BAS	Example of doing disk file handling with ZBasic (make sure "Convert to Uppercase" is set to NO under "Configure").
T 004 QUICK.APP	Quick sort. Append to SORT.BAS to see how it used.
T 003 SHELL.APP	Shell sort. Append to SORT.BAS to see how it used.
T 004 SORT.BAS	Creates random data to demonstrate the SHELL and QUICK sort subroutines above.
T 003 SIEVE.BAS	The Infamous "Sieve of Erastothenes" benchmark.
T 004 BLOAD.FUNCTION	BLOAD function you can use in your programs.
T 003 BSAVE.FUNCTION	BSAVE function you can use in your program.
T 010 BLOAD/BSAVE DEMO	Demo of the BLOAD and BSAVE function above.

APPLE DOS 3.3 APPENDIX

GETTING STARTED

1. Before doing anything... Make a Backup of the master diskette. See your DOS 3.3 users manual for instructions on using the COPYA program.
2. Put the BACKUP in a safe place.
3. You may delete all the "T" (text) files on the disk if you need more room. If you don't need "HELP", the ZBasic.HLP file may also be deleted.
4. Read the "Getting Started" section of the main reference manual.

EXECUTING ZBasic FROM DOS 3.3

There are two ways to load and execute ZBasic:

1. Put the ZBasic diskette into the first drive and turn the system on. ZBasic will BOOT automatically from the "HELLO" program.
2. After loading DOS 3.3 put a ZBasic diskette in the second and type:

```
BRUN ZBASIC <ENTER>
```

NOTE TO THE MAIN REFERENCE MANUAL

When you are reading the main reference section of this manual always take note of this icon:

It indicates a variation to this version that you will want to read (sometimes it will make note of the ProDOS version in which case you can ignore it).

APPLE DOS 3.3 APPENDIX

IMPORTANT NOTES ABOUT "MODE"

ZBasic allows you to set different graphics and text modes. This feature lets you jump from one MODE to another as your program requires.

This does introduce some unique potential problems that are easily avoided if you know about them.

1. While programs compiled in the interactive method (RUN) of ZBasic will usually operate correctly even if the MODE is not set at the beginning of a program, a program compiled to disk as a stand-alone program (RUN* or RUN+) may appear to "Hang the system" if MODE is not set.

   To solve this problem just--->**BE SURE TO SET THE MODE AT THE BEGINNING OF EVERY STAND-ALONE PROGRAM.**

2. Sometimes when typing programs in the editor, especially after pressing CTRL C or CTRL-RESET from a running program, you may experience an unresponsive screen or keyboard.

   What has happened here is that the MODE has been changed in the compiled program and needs to be reset in the editor (your keys are actually appearing on an invisible screen of another MODE). Just type:

   <RETURN> **MODE 2** <RETURN>

   Even though you will not see the keys being typed and the screen will return to normal. Do not press RESET, or REBOOT the system, as you will lose the program in memory . The above method works just fine as long as you remember that you can't the see the keys being pressed until you press <RETURN>.

CONTROL KEYS IN LISTINGS

The 80 column card responds to certain control codes. Sometimes a REM statement or quoted string may contain a control character that will set the 80 column card to 40 characters or to a different mode. Use the example above to correct the setting and delete the control character from the offending line.

APPLE DOS 3.3 APPENDIX

MISCELLANEOUS

The following is important information about how standard ZBasic commands may vary on this version.

LIST KEYS ZBasic allows you to use the cursor keys for listing programs and for use in the line editor. The delete key or the left arrow key may be used whenever the reference section says <BACKSPACE>.

FULL SCREEN EDITOR VERSUS STANDARD LINE EDITOR

This version of ZBasic does not support a Full Screen Editor. Should you desire to use a Full Screen editor you may want to try the ProDOS version of ZBasic. It comes with a Full Screen Editor built-in.

You may also use other editors to create your programs if you save your programs in ASCII using SAVE+ or SAVE*.

TEXT AND GRAPHICS

ZBASIC ALLOWS THE INTEGRATION OF TEXT AND GRAPHICS IN MODE 5 AND 7

ZBasic allows the user to integrate text and graphics on the screen just like other computers. This permits porting programs over to the Apple from the IBM PC and many other computers! (Modes 5 and 7 only.)

BLINKING CHARACTERS

Blinking Characters are not supported in Graphics mode text. Inverse characters may be obtained by setting the high bit of the character by **OR**-ing the character with 80 Hex or adding 128 to the ASCII value.

APPLE //e EXTENDED 80 COLUMN CARD CONTROL CODES

All features of the Apple™ Extended 80-column card may be used by printing characters to the screen in modes 2,4,6.

For example, printing a control-w will cause the screen to scroll up. See the ProDOS appendix for a list of the 80 column card control characters.

APPLE DOS 3.3 APPENDIX

CONTROL-RESET *VERSUS* CONTROL-C

If the computer should "Lock-Up" for some reason, or you faced with the monitor prompt (*), you can press CTRL-RESET to re-enter the ZBasic line editor. In most cases your source code will remain intact.

CTRL-RESET may also be used instead of CTRL-C to break out of programs and in many cases is preferable since it is much more responsive.

RELATIVE GRAPHIC COORDINATES *VERSUS* PIXEL COORDINATES

ZBasic provide the standard device independent graphic coordinate system of 1024x768 so that programs created on other computers will also function on the Apple // and visa-versa. Even so, should the need arise where you MUST use pixel coordinates use these POKEs:

`POKE &F388, &60`	Sets to Pixel coordinates of that MODE (see MODE chart)
`POKE &F388, &A9`	Sets back to ZBasic's standard device independent coordinates of 1024x768.

See "Graphics" in the front section of this manual for more information about using ZBasic graphics.

COMPATIBILITY WITH APPLESOFT™

ZBasic supports quite a number of the Applesoft commands. The big differences are in Graphics and Disk file handling. To see a listing of the commands that are not supported and some suggestions on converting your Applesoft programs over to ZBasic, see the Apple // ProDOS appendix in this manual.

USING WITH HARD DISKS AND/OR NON-STANDARD DOS

This version of ZBasic overwrites the area of memory that is normally occupied by the DOS 3.3 command interpreter to provide you with as much programming area as possible. ZBasic can get away with this because it does not use the command interpreter; it uses the DOS file manager directly.

Because of this, foreign operating systems (such as Diversi DOS) usually will not work with ZBasic.

In addition, hard drives that force the use of a modified DOS 3.3 may or may not work. If you need the increased speed of a compiler and require the storage capacity of a hard rive, we suggest you consider the ProDOS version of ZBasic. It does not modify ProDOS in any way and is compatible

APPLE DOS 3.3 APPENDIX

SPECIAL DOS 3.3 CONFIGURATION OPTIONS

CONFIGURING DRIVE SPECIFICATIONS

```
DRIVE  A:  SLOT  6_          DRIVE  1
DRIVE  B:  SLOT  6_          DRIVE  2
DRIVE  C:  SLOT  0_
```

Besides the regular configuration options that ZBasic offers, (See "Getting Started" in the reference section) the DOS 3.3 version allows you to configure which physical slot and drive will be associated with a logical drive specifier.

ZBasic specifies drives with letters instead of numbers (similar to MS-DOS and CP/M drive specs). D1=A, D2=B etc.

During configuration you will be prompted with a logical drive specifier e.g. A: , and asked which physical slot and drive will be associated with that logical drive spec. You are allowed to configure multiple logical specifiers for a single physical drive.

For example, you can configure both drive A: and C: to access slot 6, drive 1. You may also configure for drives that are not present on your system. You should be careful when doing this, so that you do not try to access these drives. This is however, useful when developing software for other systems when using ZBasic™. You can configure extra drives, and access them if the end users have them in their systems.

CONFIGURING PRINTER SLOTS

```
PRINTER SLOT     1-7              01    00001      ?_
```

The printer slot may also be set during configuration. This allows you to place your printer interface card in any slot. ZBasic will support any printer card that conforms to the Apple™ interface card standards for the ROMs. (Either Serial interfaces or Parallel.)

This merely sets the default printer slot. The slot may be changed at any time using the special **DEF LPRINT** statement.

The default printer slot is number 1. The Apple™ **IIc** has the equivalent of a serial printer card in slot #1.

SETTING UP A PRINTER INITIALIZATION SEQUENCE

```
ENTER THE EXACT KEYSTROKES REQUIRED FOR YOUR PRINTER AND/OR
INTERFACE CARD ("^" TO END):
```

The printer initialization string can be any sequence of up to 12 ASCII characters.

See the ProDOS appendix under "Configure" for the details of using this option (it is exactly the same syntax).

APPLE DOS 3.3 APPENDIX

APPLE DOS 3.3 APPENDIX

APPLE® // DOS 3.3 REFERENCE

This section describes the additional commands and differences to the standard ZBasic.

You will notice there are relatively few variations from the main reference section of this manual. This version of ZBasic is very good for learning programming and for creating programs that will work on virtually all versions of ZBasic with little or no changes.

When converting programs to other computers you usually only have to take COLOR, MODE, filename variations and communication device numbers into account.

APPLE DOS 3.3 APPENDIX

COLOR Statement

FORMAT **COLOR = n**

DEFINITION The same as the standard ZBasic COLOR statement. The color codes are as follows:

Mode 0: Text Characters only
Mode 2: Text Characters only
Mode 3: Same as Mode 1
Mode 4: Text Characters only
Mode 6: Text Characters only

MODE 1, 3 NUMBER	COLOR
0	Black
1	Magenta
2	Dark Blue
3	Purple
4	Dark Green
5	Grey
6	Medium Blue
7	Light Blue
8	Brown
9	Orange
10	Grey
11	Pink
12	Green
13	Yellow
14	Aqua
15	White

Mode 5: NUMBER	COLOR	*HI-RES GRAPHICS 280X192*
0	Black1	
1	Green	
2	Violet	
3	White 1	
4	Black 2	
5	Orange	
6	Blue	
7	White2	

Mode 7: NUMBER	COLOR	*DOUBLE HI-RES 560X192*
0	Black	
1	White	Note: The ProDOS version of ZBasic supports 16 colors in this mode.

REMARK The colors on the video display of Apple // computers are affected by neighboring colors and the condition of RAM. This is a phenomena of the Apple // hardware and is not correctable.

DEF LPRINT statement

FORMAT **DEF LPRINT** [=] *Slot number*

DEFINITION This command is used to configure the printer slot during runtime. After this command is used, all printer output will be diverted to the selected slot.

The slot number may be specified by a numeric expression but the value of *Slot number* MUST be between one and seven (1-7).

If value exceeds the range of 1 through 7, the system may hang up.

EXAMPLE **DEF LPRINT = 1**

REMARK Be sure to validate the slot if it is input from the user. Incorrect values may cause the system to hang up.

This command supersedes the value set under "Configure" but does not supercede the printer initialization string.

DEF MOUSE statement

FORMAT **DEF MOUSE** [=] *expression*

DEFINITION This statement defines which device (MOUSE or JOYSTICK) will be used with the ZBasic MOUSE functions.

VALUE	DEVICE
expression = zero	Apple™ Mouse Interface card in slot number 4.
expression = non-zero	Joystick/paddle port.

EXAMPLE

```
DEF MOUSE=1: REM Define as a JOYSTICK
DO
  PRINT MOUSE(1), MOUSE(2)
UNTIL MOUSE(3)
END
```

Program will print positions of the joystick
until you press the joystick button.

REMARK The default is to read a mouse card in slot 4.

Older Apple™ IIc systems have the equivalent of a mouse card in slot 4. Unfortunately, Apple changed the slot in the newer Apple //c to slot 7. If you have a newer Apple //c or want your commercial program to operate on any Apple //c, use this routine in the beginning of your program. It checks to see if it is a new or old //c board (don't use it on a //e, IIGS or Laser 128™) and configures itself accordingly.

```
LONG IF PEEK(&FBB3)=6 AND PEEK(&FBC0)=0 AND PEEK(&FBBF)=3
  POKE &D1F8, &7F
  POKE &D1FF, &7F
  POKE &D204, &7F
  POKE &D20A, &FF
  POKE &D20F, &FF
  POKE &D217, &C7
  POKE &D21C, &C7
  POKE &D21E, &70
  POKE &D222, &C7
END IF
```

Note: The ProDOS version DOES NOT require this routine.

MEM command

FORMAT MEM

DEFINITION This command is used to show the amount of memory remaining for text and object code and the amount of text and object code space used in each bank.

```
MEMORY BANK 1
   00000      Code Mem     -Code and variable space remaining.
   00000      Object       -Size of object code generated.
   00000      Variable     -Amount of variable space used.

MEMORY BANK 2
   00000      Text         -Shows amount of text space used.
   30050      Text Mem     -Amount of text room remaining.
```

EXAMPLE

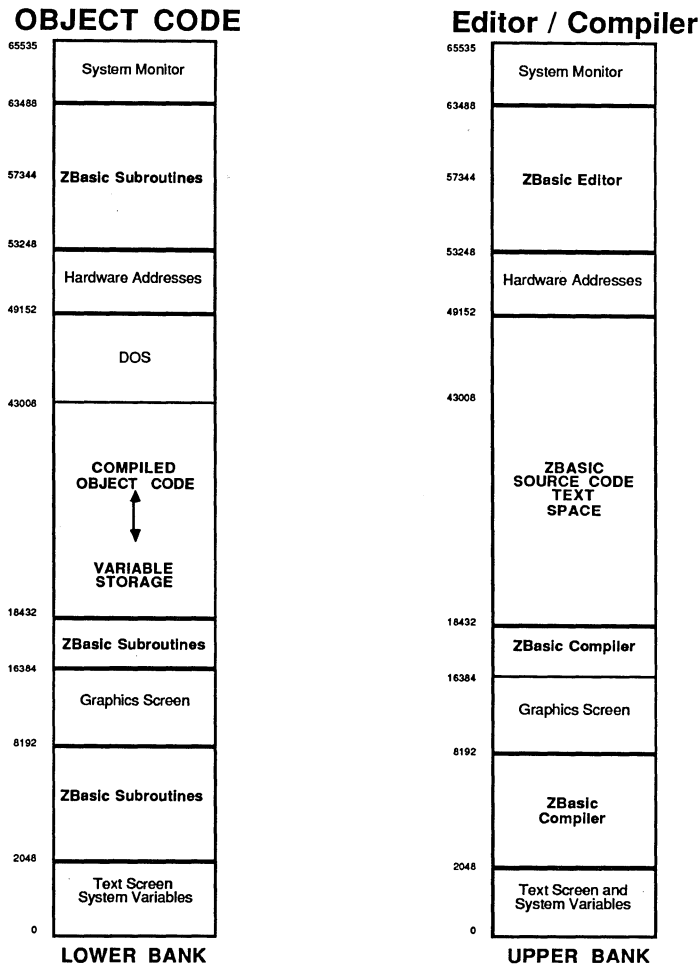
```
MEM

00043      Text
29842      Text Mem
00000      Code Mem
00000      Object
00000      Variable
```

REMARK See Memory map in this appendix.

APPLE DOS 3.3 APPENDIX

DOS 3.3 MEMORY MAP

Apple™ //e, //c Memory Map



APPLE DOS 3.3 APPENDIX

MODE statement

FORMAT **MODE** *n*

DEFINITION ZBasic uses MODE to define the characteristics of a screen.

ZBASIC ALLOWS TEXT and GRAPHICS INTEGRATION

That's right. ZBasic allows a program to integrate text and graphics anywhere on the screen in MODE 5 and 7. This feature allows ZBasic programs from an IBM PC and other computers to run on your Apple.

DOS 3.3 version MODE CHART

MODE	TEXT	GRAPHICS
0	40 x 24	character
1	none	40 x 48
2	80 x 24	character
3	none	80 x 48
4	80 x 24	character
5	40 x 24	280 x 192
6	80 x 24	character
7	80 x 24	560 x 192
MODES 8-15 are reserved for future use		

character = Text only MODE. Draws graphics using characters.
40 x 48 = Low resolution graphics
80 x 48 = Medium Resolution "Color" Graphics

280 x 192 = High Resolution "Color" Graphics
560 x 192 = Double High Resolution. Not on][+.

Modes 9, 11, 13 and 15 have graphics at the top of the screen and text at the bottom, similar to Applesoft BASIC.

APPLE DOS 3.3 APPENDIX

OPEN statement

FORMAT OPEN "--", *filenumber*, "[! filetype] [drivespec] filename", *record length*

DEFINITION This version of ZBasic has the same syntax as other versions with the exception that file specifications are within the filename:

!filetype is a number from 1 to 8 and sets the *filetype* only at the time the file is created. At all other times it is ignored. The types of files that may be defined:

!filetypes

1) Text file (default)
2) Integer BASIC file
3) Applesoft™ BASIC file
4) Binary File
5) S type file
6) Relocatable type file
7) A type file
8) B type file

See your Apple DOS 3.3 reference manual for specifics about file types.

drivespec is a letter A through H followed by a colon separator. The letter must be in upper case and specifies the physical slot and drive set in configuration.

A: D1
B: D2

filename is a standard Apple DOS™ 3.3 filename of up to 30 characters.

EXAMPLE

```
OPEN"O",1,"!4 A:FRED"
```

<---- Creates a Binary file named "Fred" on drive A: (normally D1)

REMARK Any type of file can be opened in ZBasic. If files are to be read from other software, they should be written with the correct file type and file format for that software.

If drivespec is omitted, the last accessed drive will be used as the default.

See "Files" in reference section for more information about using files. Also see the example program DISKIO.BAS on the master disk.

OPEN"C" statement

FORMAT **OPEN"C", *slot*** [, [*baud*] [, [*parity*] [, [*stopbit*] [, [*word length*]]]]

DEFINITION Same as OPEN"C" in the main reference section except that *slot* designates the slot that contains the Super Serial Card.

The default slot number is 2. The normal slot for use with a modem is slot two.

The Apple //c contains the equivalent of a Super Serial Card in slot number 2.

The IIgs must have a Super Serial Card in order to use this statement. It will not operate with the built-in IIgs serial port.

slot	
-1	Indicates Super Serial Card in slot 1
-2	Indicates Super Serial Card in slot 2
-3	Indicates Super Serial Card in slot 3
-4	Indicates Super Serial Card in slot 4
-5	Indicates Super Serial Card in slot 5
-6	Indicates Super Serial Card in slot 6
-7	Indicates Super Serial Card in slot 7

EXAMPLE

```
OPEN"C",-2,300
```

REMARK See the main reference section for more information and examples of using the OPEN "C" statement.

APPLE DOS 3.3 APPENDIX

APPLE ProDOS APPENDIX

Notes

APPLE ProDOS APPENDIX

ZBASIC™

ProDOS® 8 Version

For the Apple®][+, *//e*, *//c*, IIGS and Laser 128™

by
Greg Branche
Original DOS 3.3 version by Dave Overton

ZEDCOR, INC.

APPLE ProDOS APPENDIX

TABLE OF CONTENTS

APPLE ProDOS APPENDIX

APPLE ProDOS APPENDIX

HARDWARE REQUIREMENTS

Apple *//c*, IIGS and Laser 128

The 64K and 128K ProDOS versions of ZBasic function with a standard Apple //c and IIGS. A disk drive is required (5.25 or 3.5 inch). ProDOS provides a /RAM disk, size depending on available memory. An Apple Mouse™ with interface, Joystick and Super Serial Card are supported but are not required.

IIGS Note: The IIGS emulates the //e, //c modes with this version of ZBasic. Super High-Res graphics are not supported on this version directly.

Apple *//e*

64K Version

The 64K ProDOS version runs with a standard Apple //e. A disk drive is required (5.25 or 3.5 inch). An AppleMouse™ with interface, Joystick and Super Serial Card are supported but are not required.

If you have an Extended 80 column card, or other memory board, ProDOS provides a /RAM disk, size depending on additional memory. If you have only 64K there will be more disk accesses and compilation will take longer.

//c, //e, //GS Note: Code can be generated which will run on the older Apple][+ or][if certain restrictions are observed; Avoid MODE 3 or 7 as they require an extended 80 column card which will not function in an Apple][+.

128K Version

The 128K version of ZBasic requires an Extended 80 Column Card and a 65C02 or 65802 microprocessor.

Apple][or Apple][+

64K Version

If you have an Apple][or][+, you MUST have a 16K bank-switched memory card installed (giving you at least 64K memory) . If you have a ProDOS compatible memory board that allows ProDOS to create a /RAM disk, ZBasic will take advantage of it. If you have only 64K there will be more disk accesses and compilation will take longer.

A disk drive is required (5.25 or 3.5 inch). ZBasic requires a minimum of 64K memory to create and execute programs. An AppleMouse™ with interface, Joystick and Super Serial Card are supported but are not required.

128K Version

The 128K ProDOS version of ZBasic will not operate on an Apple][or][+.

OLDER 80 COLUMN CARDS

Older style 80 column cards may or may not function.

The Videx 80 column card works in mode 2 although you will have to do some manual switching. When typing CLS from the Standard line editor ZBasic will sense the Videx board and clear the screen automatically.

APPLE ProDOS APPENDIX

FILES INCLUDED ON THE MASTER DISKETTE
64K VERSION

The following files are included with the 64K ProDOS version of ZBasic:

File	Description
ZBASIC.SYSTEM	The boot program and low-memory subroutines.
RUNTIME.OBJ	High-memory runtime subroutines. This program MUST accompany stand-alone programs you create with ZBasic.
EDITOR.OBJ	The ZBasic command environment and Standard line editor.
COMPILER.OBJ	The ZBasic compiler.
FSEDIT.80.OBJ	80-column full screen editor for //c, IIGS and 80-col //e. May be deleted if not used.
FSEDIT.40.OBJ	40-column full screen editor for 40-col //e ,][and][+. May be deleted if not used.
INIT.64.OBJ	Contains a stand-alone program initialization sequence.

THE FILES ABOVE ARE REQUIRED WHEN CREATING ZBASIC PROGRAMS.

THE FILES BELOW ARE OPTIONAL OR EXAMPLE PROGRAMS

ZBASIC.HLP	Help file accessed with the "HELP" command.
DISKIO.BAS	Sample program demonstrating ZBasic file commands.
GRAPH.BAS	Sample program demonstrating ZBasic graphics.
SORT.BAS	Program to illustrate the use of the QUICK.APP and SHELL.APP sorting programs. Load this program first then type: APPEND 1000 QUICK.APP (or SHELL.APP).
QUICK.APP	Append file containing a quicksort subroutine.
SHELL.APP	Append file containing a shell sort subroutine.
SIEVE	The SIEVE benchmark program from BYTE magazine.
GRAPHICS.COLORS	Demonstrates the colors available in each of the graphics modes.
BLOAD.SAMPLE	Demonstrates the use of the BLOAD and BSAVE functions.
BSAVE.FN	Function to simulate the ProDOS BASIC.SYSTEM BSAVE command.
BLOAD.FN	Function to simulate the ProDOS BASIC.SYSTEM BLOAD command.
DHRBSAVE.FN	Double Hi-Res BSAVE function saves Double Hi-Res Graphic screen.
DHRBLOAD.FN	Double Hi-Res BLOAD function loads Double Hi-Res Graphic screen.
DRAW.FN	Function to simulate the Applesoft DRAW command.
PREFIX.SAMPLE	Sample program demonstrating the use of the PREFIX function.
PREFIX.FN	Function to set or retrieve the ProDOS default prefix at runtime.
CREATE.FN	Function to create a ProDOS subdirectory from within a ZBasic program.
DATETIME.FN	Function to manually set the date and time.

APPLE ProDOS APPENDIX

FILES INCLUDED ON THE MASTER DISKETTE
128K VERSION

The following files are included on the 128K ProDOS version of ZBasic (flip side of the diskette):

File	Description
ZBASIC.SYSTEM	The boot program and low-memory subroutines.

*The following **three** files MUST accompany stand-alone programs you create with ZBasic.*

RT.MAIN.OBJ1	High-memory runtime subroutines.
RT.AUX.OBJ0	Low auxiliary memory routines.
RT.AUX.OBJ1	High auxiliary memory routines.
EDITOR.OBJ0 EDITOR.OBJ1 EDITOR.OBJ2	The ZBasic command environment and Standard line editor and Full Screen Editor.
COMPILER.OBJ0 COMPILER.OBJ1	The ZBasic compiler.
INIT.128.OBJ	128K Stand-alone program initialization sequence.

Use the example programs on the 64K side of the diskette (see previous page for details). There will also be a couple examples on this side of the diskette. These programs will not work with the 64K version.

APPLE ProDOS APPENDIX

GETTING STARTED

1. Make a BACKUP of your master ZBasic diskette. Store the master in a safe place (refer to the ProDOS reference manual for backup methods).

   Note: There are two versions of ZBasic for ProDOS; a 64K version and a 128K version. On 5.25" diskettes they occupy opposite sides. On 3.5" diskettes they are in two different sub-directories. If using 5.25" diskettes make sure to backup both sides.

2. Due to storage limitations, the ZBasic disk does not contain the ProDOS operating system. Therefore you must create a ProDOS environment. There are a couple of ways to do this:

   a. **BOOT FROM A ProDOS Master Disk** (/USERS.DISK). Then type "B" from the menu to enter Applesoft BASIC.

      From the prompt ( ]), enter: "PREFIX /ZBASIC", then : "-ZBASIC.SYSTEM".

   b. **CREATE A ZBASIC BOOT DISK**: Format a blank disk (using the FILER utility). Copy the file "PRODOS" from a ProDOS disk to your freshly formatted disk. Transfer the following files from your ZBasic Master Disk to your new copy:

      64K VERSION

      ```
      ZBASIC.SYSTEM          EDITOR.OBJ
      RUNTIME.OBJ            INIT.64.OBJ
      COMPILER.OBJ
      FSEDIT.80.OBJ  (use FSEDIT.40.OBJ if using a 40 col Apple ][, ][+, or //e)
      ```

      128K VERSION

      ```
      ZBASIC.SYSTEM          COMPILER.OBJ0
      RT.MAIN.OBJ1           COMPILER.OBJ1
      RT.AUX.OBJ0            INIT.128.OBJ
      RT.AUX.OBJ1            EDITOR.OBJ0
      EDITOR.OBJ1            EDITOR.OBJ2
      ```

      CTRL <OPEN APPLE> RESET will now load and execute ZBasic from this disk.

3. Read this appendix, making notes of any variations.

4. Now read "Getting Started" in the main reference section.

APPLE ProDOS APPENDIX

BOOT-UP PROCESS

When the ZBASIC.SYSTEM program is loaded from ProDOS, it does several things prior to putting you into the editor:

- ZBasic Title page displayed during the boot process.
- Zero page locations are initialized.
- The low-memory runtime module is moved into place.
- ZBasic looks for a volume with the first four characters "/RAM". If found, it will copy the necessary system files into the ram disk. If you do not wish to have ZBasic use the /RAM disk, simply rename it prior to loading ZBasic.
- The command environment and standard line editor overlay is loaded into memory (to invoke the full screen editor type `EDITOR` or `EDITOR+`).

NOTE TO THE MAIN REFERENCE SECTION

Wherever there are notable differences between the text and the Apple ProDOS version you will see an Apple ICON that will tell you the difference or refer you to the correct section. The icon looks like this:

Occasionally the icon refers to the Apple // DOS 3.3 version. In those instances simply ignore this icon.

THE IMPORTANCE OF USING A RAM DISK

In order to leave as much free memory as possible for program development , there is a lot of overlay swapping and other disk access involved while editing and running a program interactively (like an interpreter).

For example, if you type "PRINT 2.345*32" from the editor command line, quite a number of events take place:

- the editor saves whatever program you have in memory to the disk (/RAM disk, if enabled).
- loads and runs the compiler from disk (/RAM disk, if enabled).
- compiles the command and stores the object code in memory.
- loads and runs the runtime system (/RAM disk, if enabled).
- the runtime executes the object code (which in this example prints 75.04).
- then reloads and executes the editor (/RAM disk, if enabled).
- editor reloads the temporary file (/RAM disk, if enabled) and waits for the next command.

Phew! As I said, a lot of disk access! It should be obvious that a /RAM disk will speed up the whole process 10-15 times since disk access is nearly eliminated.

APPLE ProDOS APPENDIX

USING THE RAM DISK

These versions of ZBasic require 64K and 128K of memory, respectively. If your system has more then the minimum amount of memory required, and the extra memory is configured as a ProDOS /RAM disk, ZBasic will use it to store some system files and overlays so that overall program development time will be reduced and system speed will be improved.

In addition, a temporary file used to hold your source code is also saved to disk during the overlay swapping. This file is named ZTEMP.ZBS.

If there is no /RAM volume, the ZBasic disk MUST remain in the drive for normal operation.

If the /RAM disk is not large enough to hold ZTEMP.ZBS, ZBasic returns a DISK FULL error and returns to the editor. You should save the file to a diskette and compile from disk at this point (RUN*) or exit ZBasic, disable the /RAM disk by renaming it from ProDOS, then re-enter ZBasic without rebooting.

Warning: DO NOT RENAME THE /RAM DISK WHEN IN USE!

ProDOS PATHNAMES

The filenames used in ZBasic are standard ProDOS pathnames. ProDOS pathnames can consist of up to 64 characters, including separating slashes. Individual filenames can be up to 15 characters long, and can consist of alphanumeric characters and periods only.

Pathnames may be used with OPEN, RENAME, SAVE, LOAD and all other disk commands and statements. See your ProDOS manual for more information about pathname syntax.

FILE BUFFER SIZE--- OR HOW TO GET AN EXTRA 2048 BYTES

Each file opened by a ZBasic program requires a 1024 byte file buffer. ZBasic defaults to two file buffers (2048 bytes).

If you configure ZBasic for one file buffer, 1024 bytes is freed for program or variables (configuring for no open files would free 2048 bytes).

See "Configure" in main manual.

APPLE ProDOS APPENDIX

LIST KEYS

The following is a list of additional keys which can be used in the command mode editor to list lines of source code (as well as those described in the main manual):

Up Arrow	List previous line
Down Arrow	List next line
Left Arrow	List first line of the file
Right Arrow	List last line of the file

HELP

The file used by the HELP command is named "ZBASIC.HLP". If you so desire, this file can be deleted to allow more storage room on the disk. If ZBasic is not able to find this file in it's system directory, it will look in the user's currently logged directory (see the PATH command). If ZBasic still cannot find the help file, you will get a "File-Not-Found" error.

CONTROL-RESET *VERSUS* CONTROL-C

This version allows you to use either CTRL-C or CTRL-RESET to exit a running program. If the computer should "lock up" for some reason, or you are faced with the monitor prompt (*), you can press CTRL-RESET to restart the ZBasic editor. Your source program should remain intact. If you press CTRL-RESET while executing a stand alone program, the program will be terminated, and you will be allowed to load and execute another ProDOS system program. If you are faced with the monitor prompt anywhere within the ZBasic system, pressing CTRL-Y will also return you to the editor.

ADDITIONAL DISK ERROR CODES

In addition to the standard ZBasic disk error codes referred to in the main reference section, the following codes are defined and may be trapped with ON ERROR GOSUB:

Error Code	Error Message
9	Position Error
10	No Device Connected Error
11	Disk Switched Error
12	Duplicate Filename Error
13	Incompatible File Format Error
14	Access Error
15	File Already Open Error
16	Directory Structure Damaged Error
17	Not a ProDOS Volume Error
18	Duplicate Volume Online Error
19	File Structure Damaged Error
20	I/O Error
21-255	Disk Error

The actual disk error code will be the filenumber times 256 plus the number above. See disk error in the main reference manual for more information.

APPLE ProDOS APPENDIX

HEXADECIMAL CONSTANT INDICATORS ($ and &)

In addition to the "&" prefix signifying a hexadecimal constant (as in &FF69), the "$" character may also be used (as in $FF69). This is so that Apple users will feel more at home. Remember that if this character is used the program will not be directly transportable to the Apple DOS 3.3, IBM, Macintosh, CP/M, or other versions of ZBasic.

RELATIVE GRAPHIC COORDINATES VERSUS PIXEL COORDINATES

The standard ZBasic graphic coordinate system is great for porting programs between the various computers that run ZBasic or between Hi-Res and double Hi-Res. Occasionally you may need to switch to PIXEL coordinates. Use this statement:

POKE WORD &85, 0

After this statement is executed, the following screen dimensions will be in effect with the different graphics modes:

MODE 1 40 x 40
MODE 3 80 x 40
MODE 5 280 x 192
MODE 7 560 x 192 (not available with the Apple][+ or //e without an extended 80 col. card)

MODE automatically resets to the device independent coordinate system, so you must use the POKE WORD &85, 0 statement immediately after setting MODE to re-enable the pixel coordinates above.

MOUSE

If your program uses the MOUSE function to receive input from the mouse (DEF MOUSE=0), you MUST use a MOUSE(0) function at the beginning of the program prior to any other MOUSE call.

MOUSE(0) forces ZBasic to scan the slots for a mouse interface card, and then initialize the mouse properly. If the mouse is not initialized prior to accessing it, your program may die a horrible death (crash)!

In addition, if a mouse interface could be found and initialized properly, MOUSE(0) will return a value of -1 (true,) otherwise a value of zero (false) will be returned.

APPLE ProDOS APPENDIX

IMPORTANT NOTES ABOUT VIDEO/SYSTEM PROBLEMS

ZBasic allows you to set many different graphics and text modes. This feature lets you jump from one MODE to another as your program requires. This does introduce a unique potential for confusing video problems that are easily mistaken for system errors.

• While programs compiled in the interactive method (RUN) of ZBasic will usually operate correctly even if the MODE is not set at the beginning of a program, a program compiled to disk as a stand-alone program (RUN* or RUN+) may appear to "Hang the system" if MODE is not set. To solve this problem; BE SURE TO SET THE MODE AT THE BEGINNING OF EVERY STAND-ALONE PROGRAM. If using an Apple][+ (or //e without an extended 80 column card) be sure to avoid MODE 3 and 7.

• Sometimes when typing programs in the editor, especially after pressing CTRL-C or CTRL-RESET from a running program, you may experience an unresponsive screen or keyboard. Nine times out of ten what has happened here is that the MODE has been changed in the compiled program and needs to be reset in the editor (your keys are actually appearing on an invisible page of another MODE). Just type:

```
<RETURN> MODE 2 <RETURN>
```

Even though you will not see the keys being typed, the screen will return to normal when you're finished typing. Do not REBOOT the system, as you will lose the program in memory. Remember: You can't see the keys being pressed until you press <RETURN>.

• CONTROL KEYS IN LISTINGS: The 80 column card responds to certain control codes. Sometimes a REM or quoted string may contain a control character that may set the 80 column card to 40 characters or to a different mode. Use the example above to correct the setting and delete the control character from the offending line.

USING THE SUPER SERIAL CARD

The file number specified in serial I/O must be the negative slot # in which an Apple Super Serial Card is installed. The Apple IIc has the equivalent of a Super Serial Card installed in slot # 2. This card would be accessed by:

```
OPEN "C",-2,300...
```

ZBasic communication commands only support the Apple Super serial card and compatible serial interfaces.

Note: The IIGS serial port is not yet supported. A Super Serial Card or compatible card or modem will function properly.

COMMANDS NOT SUPPORTED IN THIS VERSION OF ZBASIC

The following two functions are not supported: **INP( )** and **OUT( )** . See the notes at the bottom of the pages in the main reference section for commands that may not be fully compatible.

APPLE ProDOS APPENDIX

INTEGRATION OF TEXT AND GRAPHICS

Unlike Applesoft, ZBasic allows you to integrate text and graphics on the screen.

This permits porting programs over to the Apple from the IBM PC and many others (MODE 5 only on the][+ and 64K //e. MODE 5 and 7 only on the 128K //e and //c).

80 COLUMN CARD CONTROL CODES

The Apple 80-column text card firmware supports many control codes to perform special operations, such as screen scrolling up and down. These control codes are available in modes 2 and 6. Simply print CHR$(x), where x is the code for the function you want to perform (see the 80-column text card manual for a listing of these codes.)

In addition, several of these codes are available in Modes 5 and 7. The codes and the function they perform in MODES 5 and 7 are listed in the following table:

CHR$ Code	Function
7	Beep the Apple speaker
8	Moves cursor position one space to the left; from left edge of window, moves to right end of line above
10	Moves cursor position down to next line, scrolls if necessary
13	Moves cursor position to left end of next line, scrolls if necessary
14	Sets display format normal (white on black)
15	Sets display format inverse (black on white)
22	Scrolls the display down one line, leaving the cursor at the current position.
23	Scrolls the display up one line, leaving the cursor at the current position
24	Turns MouseText off
27	Turns MouseText on
28	Moves cursor position one space to the right; from right edge of window, moves it to left end of line below

Other Apple screen control codes are not implemented (as control codes) for graphics MODE 5 and 7.

INVERSE TEXT

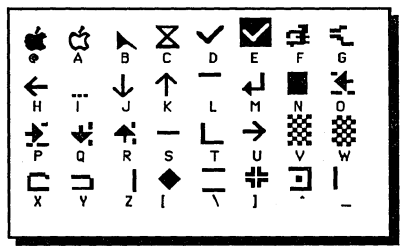
To shift to the inverse character set, print a CHR$(15). All characters printed after this will be in inverse text.

To switch back to normal text, print a CHR$(14). These are the same control codes that Apple's 80-column card uses to switch modes. As mentioned before, this works with the 40-column screen also (a slight enhancement to Apple's firmware done by our software).

APPLE ProDOS APPENDIX

MouseText CHARACTERS

In addition to the MouseText characters available in 40 and 80 column modes of the new Apple // machines, MouseText is available in Modes 5 & 7. To shift the character set to MouseText, print a CHR$(27) and a CHR$(15).



To de-select MouseText, print a CHR$(14) and a CHR$(24). Since Apple's procedure for printing MouseText requires you to shift to inverse mode (the CHR$(15)), you might think that inverse MouseText isn't possible. Not so with ZBasic! If you want to experiment a little, just use a CHR$(27) to select inverse MouseText, and CHR$(24) to select normal alphanumerics again!

CUSTOM CHARACTER SETS

The character set that is included with your ZBasic system and used by the graphics character driver is the standard ASCII character set with the addition of the MouseText characters (MODE 5 and MODE 7 only).

If you wish, you can customize the character set to your liking. Space does not permit getting into the specifics of how each character is defined or used, but I can tell you how to change the character set to a pre-defined set. Our character set is defined in exactly the same way as the character sets included on the DOS Toolkit disk, available from Apple Computer, Inc. To change the character set, follow these instructions:

1. From Applesoft BASIC, with ProDOS active, insert a **BACKUP COPY** of your ZBasic master disk in the drive.

2. Type: *64K:* `BLOAD "/ZBASIC/ZBASIC.SYSTEM, A$2000,TSYS"`
   128K: `BLOAD "/ZBASIC/RT.AUX.OBJ0, A$2800"`
   This loads the character set (and some other stuff) into memory.

4. Load your character set by typing:
   `BLOAD <your character set pathname>, A$3900"`
   This loads your character set over our character set. Since the DOS Toolkit character sets are only 768 bytes long, (characters 32-128) and only contain definitions for the standard ASCII characters, you will not be overwriting the MouseText (0 to 31).

5. Re-insert your ZBasic master disk, and type:
   64K: `BSAVE "/ZBASIC/ZBASIC.SYSTEM, A$2000,L$4000,TSYS"`
   128K: `BSAVE "/ZBASIC/RT.AUX.OBJ0, A$2800"`

APPLE ProDOS APPENDIX

SPECIAL ProDOS CONFIGURATION OPTIONS

ZBasic can be configured by typing "C" at the initial prompt screen (see the "Configure" section of the main reference manual), or by typing "CONFIG" while in the editor (see "CONFIG" on the next page). In addition to the standard configuration parameters, there are two more parameters which you can set for the Apple //.

```
PRINTER SLOT? 1-7
```

This allows you specify which slot your printer interface is in. This number must be from 1 to 7 (slot 1 is the standard printer slot for Apples). As in the rest of the configuration questions, pressing <RETURN> as a response will accept the default and skip the initialization string configuration.

If you type a number from 1 to 7, you are telling ZBasic that your printer interface card is in that slot and you will be given an opportunity to specify a printer initialization string (the //c has the equivalent of an Apple Super Serial Card in slot 1):

SETTING UP A PRINTER INIALIZATION SEQUENCE

```
ENTER THE EXACT KEYSTROKES REQUIRED BY YOUR
PRINTER AND/OR INTERFACE CARD ("^" TO END):
```

The printer initialization string can be any sequence of up to 12 ASCII characters that can be typed from your keyboard (end input with the "^" symbol (caret).

To enter the initialization string, type the EXACT keys required by your printer and/or interface card. The keys will appear on the screen as you type them. Unprintable control characters will appear prefixed by a caret(^) character on the screen. Once set, this string is sent to the printer prior to anything else being sent out (such as LLIST, LPRINT, or ROUTE 128). Be sure to see the <S>ave option under "Getting Started" in the front of this manual.

Some common control codes may be entered from the keyboard using:

CTRL H	=8	TAB or CTRL I	=9	CTRL J	=10
CTRL L	=12	RETURN	=13	DELETE	=127
CTRL \	=28	CTRL]	=29	CTRL ^	=30
CTRL _	=31	ESC or CTRL [	=27		

See your Apple reference manual for other character sequences.

This is most useful for those users who have an older interface card that does not interface correctly with the 80-column screen. These cards will echo characters to the screen using the 40-column screen firmware, instead of the 80-column firmware when the 80-column card is active (usually messing everything up).

One solution is to tell the interface card to NOT echo characters by using the following initialization sequence: `<CONTROL I>80N`

This would instruct the interface to turn off the screen, and allow up to 80 characters per line on the printer. See your interface card manual for more details. You can also send printer configuration characters to your printer for all kinds of fancy printing, if your printer is capable of it. Your printer manual will list printer control codes that are applicable.

continued...

APPLE ProDOS APPENDIX

continued from previous page

```
LOCATE order X,Y?          <Y/N>          Y_
```

This option allows you to configure the order of the coordinates in the LOCATE statement.

Normally ZBasic expects the horizontal (X) coordinate first. By answering "N" to this question you can make the vertical (Y) coordinate first and the horizontal (X) coordinate second.

Note: This also alters the coordinate base of the screen to make the upper-left hand corner character position 1,1 instead of 0,0 (only affects LOCATE).

This option is provided to maintain compatibilty with the IBM/MSDOS versions of ZBasic which have this option so that BASICA programs are easier to convert. This makes porting BASIC programs from other computers much easier.

CONFIG

You may re-configure the system any time from the standard line editor by using the CONFIG command. Use caution when doing this while working with CHAIN programs or programs that will be sharing data (especially floating point numbers).

Each CHAINed program must be compiled using the same configuration as the other programs in the overall CHAINed system. Otherwise, you will get a chain error when attempting to run them.

If you elect to (S)ave your custom configuration, ZBasic will ask you to enter the complete pathname of the ZBASIC.SYSTEM file. This will normally be "/ZBASIC/ZBASIC.SYSTEM", unless you have installed ZBasic on a hard disk and/or changed the name of this file.

If ZBasic has trouble saving your configuration, it will give you an error message and wait for a keypress. After pressing a key, you will be returned to the configure menu.

If no error is encountered, you will automatically be put into the line editor.

NOTE ON CASE CONVERSION

During boot-up, the system checks to see if it is running on an Apple][+ or a newer machine. If you are using a][+ the system will automatically convert from lower to upper case for both keyboard input and screen output.

If it's a newer machine, the system will skip the conversion. Upper/lower case conversion can be configured separately by the user from the configure menu. See "Configure" in the front of this manual.

APPLE ProDOS APPENDIX

THE ProDOS MACHINE LANGUAGE INTERFACE

These versions of ZBasic have been written with the ease of direct access to ProDOS in mind. This section of the manual describes how a ZBasic program can talk with ProDOS directly.

MLI INTERFACE

First of all, this is NOT a tutorial on how to use the ProDOS Machine Language Interface. For more information on that subject, consult the ProDOS Technical Reference Manual.

ENTRY POINTS

All parameters for ProDOS calls that are made by ZBasic are located in a parameter block at $1F00 (all addresses are in HEX). There is an 18 byte buffer here that the ZBasic system uses for all MLI calls (18 being the maximum length of any MLI parameter block).

In addition, the entry point for a ProDOS call with the 64K version is at $803 ($865 for the 128K version). One more buffer that might be useful is the file name buffer. It is located at $1F12, and is 64 bytes long (the maximum length of a ProDOS pathname).

ZBasic TO ProDOS INTERFACE

To use the ZBasic to ProDOS interface, first set up the parameter list for the MLI call that you wish to make. If you need to, you can set up the pathname pointer with:

```
POKEWORD &1F01, VARPTR(name$)
```

since ZBasic strings conform to the ProDOS pathname standards (a count byte followed by the string). Next, you must load the 6502 Accumulator with the MLI command code, and then JMP or JSR to location $803 ($865 for the 128K version). The ProDOS call will be performed, and the carry flag will have the status of the call upon return. If the carry is clear, then the call returned with no error. If, on the other hand, the carry is set, then there was an error and the error code can be retrieved from location $A2. The ProDOS error that is returned by the MLI is translated into the appropriate ZBasic error code, if possible. If not, then the actual MLI error code will be returned. If you wish to use the standard ZBasic error handler, then you can perform a JMP to location $809 ($87F for the 128K version) if the carry is set upon return from the ProDOS interface.

For example, to use this interface to set the ProDOS system prefix to "/ZBASIC":

```
PATH$ = "/ZBASIC"
POKE &1F00, 1                    <--- Sets up parameter block
POKE WORD &1F01, VARPTR(PATH$)
MACHLG &A9, &C6 &20, $803        <--- 128K version change to $865
MACHLG &90, 3, &20, $809         <--- 128K version change to $87F
END
```

Note: Either "&" or "$" may be used to denote Hex numbers (ProDOS version only).

APPLE ProDOS APPENDIX

USING MACHLG

The assembly language source for the MACHLG statements would look something like this:

```
SET.PREFIX      EQU     *
                LDA     #$C6    ;MLI CODE FOR SET_PREFIX
                JSR     $803    ;CALL THE INTERFACE
                BCC     DONE    ;NO ERROR
ERROR           JSR     $809    ;LET ZBASIC HANDLE THE ERROR
DONE            EQU     *
```

ProDOS ERROR CODES

For the 64K version only; another location of interest is $806. This is the entry point for the subroutine that translates ProDOS error codes into ZBasic error codes. If you wish to access the MLI directly, but still want ZBasic error codes returned, you can perform a JSR to this subroutine with the ProDOS error code in the accumulator. The translated error code will be stored in location $A2.

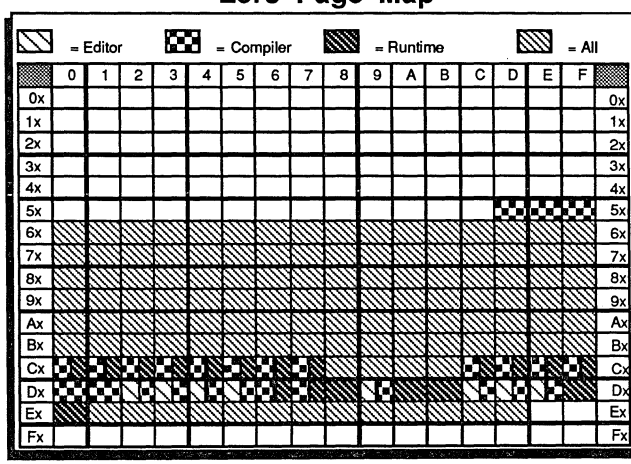
For more examples of how to use the ProDOS-ZBasic interface, see the CREATE, PREFIX, BLOAD, and BSAVE functions included on your master diskette.

MEMORY USAGE

The following diagrams illustrate memory usage for the various phases of operation of the ZBasic system.

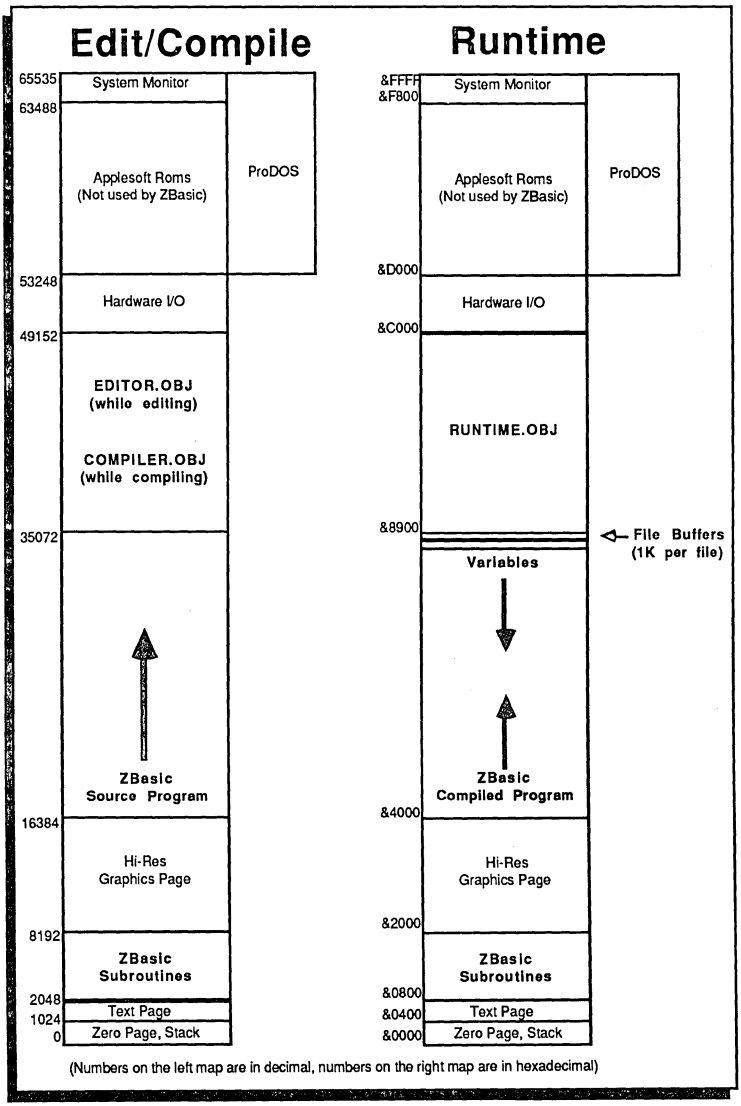
Note: Memory locations 768-975 (page 3) are not used by the ZBasic system. This would be a good place to store short machine language subroutines.

Zero Page Map



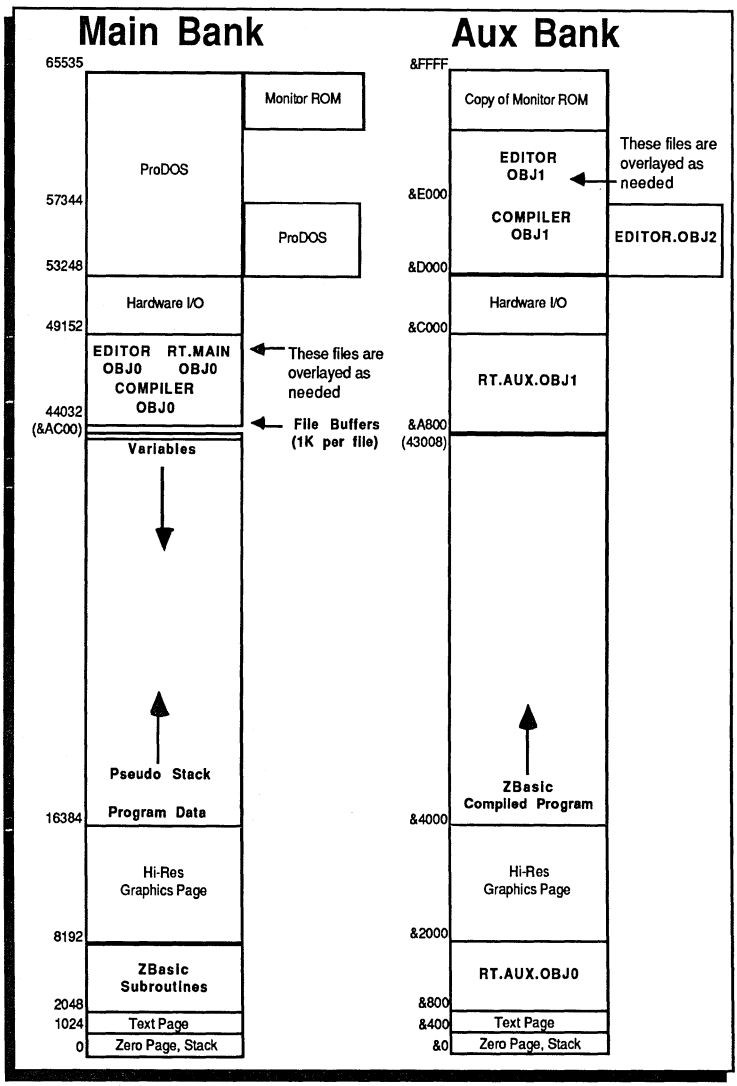
APPLE ProDOS APPENDIX

64K VERSION MEMORY MAP



APPLE ProDOS APPENDIX

128K MEMORY MAP



APPLE ProDOS APPENDIX

CONVERTING APPLESOFT PROGRAMS TO WORK WITH ZBASIC

ZBasic is an advanced version of BASIC. While it shares many of the commands and syntax of Applesoft, it is not exactly the same in many areas, such as graphics, disk file handling and such.

CONVERTING APPLESOFT FILES FOR LOADING INTO ZBASIC.

ZBasic source code files and Applesoft files are not compatible. To convert an Applesoft program so you can load it into ZBasic:

1. Make sure you have a Backup of your Applesoft program then load the Applesoft program into Applesoft. Make sure your program doesn't have a line zero then add the following line to the program.

```
ØF$="FILENAME":PRINTCHR$(4)"OPEN";F$
:PRINTCHR$(4)"WRITE";F$:POKE33,33:PRINT"Ø";
:LIST 1-:PRINTCHR$(4)"CLOSE":TEXT:END
```

   Note: The program above is one line. Enter without spaces or <RETURN>.

2. Change "FILENAME" above to the name of the file you wish to create for loading into ZBasic. Then type RUN.

3. Load ZBasic, press "E" for edit, and then load the program using LOAD. To compile the program type RUN. When errors occur use the chart on the next few pages to convert syntax to ZBasic syntax.

CONFIGURING ZBASIC FOR COMPATIBILITY WITH APPLESOFT

ZBasic allows you to configure the system for your preferences. To make ZBasic as compatible as possible to Applesoft, set the following configurations. See "Configure" in the front of this manual for details about setting configuration options:

Default Variable type:	**S** (avoid doing this whenever possible)
Convert to Uppercase Y/N:	**Y**
Optimize Expressions for integer Y/N:	**N** (avoid doing this whenever possible)

STRING LENGTH NOTE

ZBasic uses strings differently than Applesoft. See "Converting Old Programs" and DIM and DEF LEN in the front section of this manual for more information.

COMMANDS THAT ARE DIFFERENT

The following commands are different and will require converting.

The list includes hints on how to convert the various Applesoft statements to ZBasic equivalents.

APPLE ProDOS APPENDIX

Applesoft Commands	ZBasic Equivalent
BLOAD/BSAVE	See the BLOAD and BSAVE functions on the master disk.
CALL	ZBasic uses a constant as an address (not a variable). Parameters not allowed.
CLEAR	See CLEAR in the main reference section for ZBasic's additional options.
COLOR	ZBasic uses this statement for all graphics modes (not just low-res).
CONT	Not supported (ZBasic is a compiler).
DEF FN	More options in ZBasic. See DEF FN and LONG FN in the main reference section.
DIM	ZBasic only allows constants in DIM exrpressions. See DIM.
DRAW	Not available (see DRAW.FN example on the master disk).
FLASH	Not available.
FRE	Not applicable (and not necessary since ZBasic doesn't do "Garbage collection").
GET	Not available. See GET.FN on the master disk.
GR	Use: MODE 1:CLS.
HCOLOR	Use: COLOR.
HGR	Use: MODE 5 (also see MODE 7 for double hi-res).
HGR2	Not applicable
HIMEM	Not applicable
HLIN	Use PLOTx,y TO x2,y2
HOME	Use CLS.
HPLOT	Use PLOT
HTAB	Use LOCATE x, PEEK(37) (also see PRINT@ /% and INPUT@/%)
IN#	Use INSLOT
INVERSE	Use CHR$(15). See "Inverse Characters" in this appendix.
LOMEM	Not applicable
NORMAL	Use CHR$(14). See "Inverse Characters" in this appendix.
ON ERR GOTO	See ZBasic's ON ERROR GOSUB statement.
PDL	See DEF MOUSE and MOUSE in this appendix and the main reference section.
POP	Use RETURN nnnn instead.
POS(expr)	Expr=0 for default device, 1 for printer and 2 for disk.
PR#	See OUTSLOT, LPRINT, OPEN"C" and ROUTE.
RECALL	Not available.
RESUME	Use RETURN with ON ERROR GOSUB
ROT	Not available.
RUN	See RUN in this appendix and in the main reference section for other options.
RND (n)	ZBasic returns an integer number between one and n.
SCRN	Use POINT
SCALE	Not available.
SHLOAD	Not available.
SPEED	Not available.
STORE	Not available.
TEXT	Not available. Use MODE 0, 2, 4 or 6 instead. See MODE.
TRACE	Use TRON or TROFF (also see TRONX, TRONS).
VLIN	Use PLOT x,y TO x2,y2
VTAB y	Use LOCATE PEEK(36), y (see PRINT@ /% and INPUT@/%)
WAIT	Not available.
XDRAW	Not available.

Many of the commands Applesoft supports have extentions in ZBasic. For instance; ELSE is supported with IF THEN. RESTORE will allow you to position the DATA pointer to a specific item, PRINT USING is supported, etc.

Note: When converting programs a word processor with FIND and REPLACE is very handy.

continued...

APPLE ProDOS APPENDIX

DIFFERENCES IN DISK FILE COMMANDS

Applesoft File Commands	ZBasic Equivalents
OPEN A FILE FOR INPUT	
PRINTCHR$(4)"OPEN filename"	
PRINTCHR$(4)"READ filename"	OPEN"I",filenum, "filename"
OPEN A FILE FOR OUTPUT	
PRINTCHR$(4)"OPEN filename"	
PRINTCHR$(4)"WRITE filename"	OPEN"O",filenum,"filename"
OPEN A FILE FOR READ/WRITE	
PRINTCHR$(4)"OPEN filename, L100"	OPEN"R",filenum,"filename",100
PRINTCHR$(4)"OPEN filename, R10"	RECORD#filenum,10
CLOSE FILES	
PRINTCHR$(4)"CLOSE filename"	CLOSE#filenumber
PRINTCHR$(4)"CLOSE"	CLOSE

Note: Also see "Files" in the front section of this manual for more information about ZBasic's powerful file handling commands. Also see: RECORD, READ#, WRITE#, DIM, PRINT#, INPUT# and LINEINPUT#.

PEEKS, POKES, AND SYSTEM CALLS

Applesoft	ZBasic Equivalents
CALL -958	CLS PAGE
CALL -868	CLS LINE
X=PEEK(-16336)	See SOUND in reference section.
X=PEEK(-16287), Y=PEEK(-16286)	See MOUSE(3) and DEF MOUSE

Other PEEK and POKE statements should work as expected except those dealing with Applesoft.

Also see MACHLG, USR, CALL and LINE in the main reference section.

APPLE ProDOS APPENDIX

TRANSFERRING ZBasic DOS 3.3 FILES TO THE ProDOS VERSIONS OF ZBasic

The file format for the ProDOS version of ZBasic is different than the file format for the DOS 3.3 version of ZBasic. Therefore; follow these instructions to convert files:

1. LOAD your program into the DOS 3.3 version of ZBasic.
2. Use the SAVE* command to save the source code in ASCII.
3. Exit the DOS 3.3 version of ZBasic and boot your favorite DOS 3.3 to ProDOS conversion program; such as *CONVERT* found on your ProDOS /USERS.DISK or *APPLE SYSTEMS UTILITIES*.
4. Copy the file just saved in ASCII to a ProDOS formatted diskette.
5. Execute either the 64K or 128K ProDOS versions of ZBasic and LOAD the program.

Programs created in the DOS 3.3 version should run with few, if any, changes; although you may want to modify the programs to take advantage of the ProDOS /RAM disk or the 16 colors available in Double Hi-Res.

APPLE ProDOS APPENDIX

REFERENCE SECTION

This section of the appendix discusses commands unique to the ProDOS version of ZBasic and commands that may have other meanings other than those described in the main reference section.

APPLE ProDOS APPENDIX

CLS command

FORMAT **CLS** [*n*]

DEFINITION Clears the screen. Same as the standard CLS statement with the following variations:

- If you are currently editing in one of the text modes, the screen will be cleared immediately (without going through the compiler).
- If you are currently in one of the graphics modes, the command must first be compiled before it is executed by the runtime system, and takes longer.

EXAMPLE See CLS in the main reference section for detailed information.

REMARK This command works correctly with the standard Apple 80-column card and Videx 80-column cards (and compatible). Any control key typed at the keyboard that is not defined as an editor command will be passed through unchanged to the 80-column firmware. What this means is that if your card requires a CHR$(26) to clear the screen you can press CTRL-Z to accomplish the same thing.

To use any of the other CLS options from within the editor, such as CLS nn, precede the command with a colon. e.g. `:CLS ASC("A")`

APPLE ProDOS APPENDIX

COLOR statement

FORMAT **COLOR** [=] *n*

DEFINITION The **COLOR** codes for the ProDOS version of ZBasic are:

Modes 0, 2, 4, & 6: Text Characters only, no color.

Modes 1, 3, & 7:

NUMBER	COLOR
0	Black
1	Magenta
2	Dark Blue
3	Purple
4	Dark Green
5	Grey
6	Medium Blue
7	Light Blue
8	Brown
9	Orange
10	Grey
11	Pink
12	Green
13	Yellow
14	Aqua
15	White

Mode 5:

NUMBER	COLOR
0	Black1
1	Green
2	Violet
3	White 1
4	Black 2
5	Orange
6	Blue
7	White2

IIGS Note: The IIGS Super Hi-Res graphics mode is not supported directly (the //e, //c modes are emulated).

APPLE ProDOS APPENDIX

DATE$, TIME$ function

FORMAT **DATE$**
TIME$

DEFINITION See the main reference manual.

EXAMPLE See the main reference manual for details of usage.

REMARK These functions behave exactly as described in the standard reference section if your system has a ProDOS compatible clock installed.

The system performs a ProDOS call to retrieve the date and time from a clock card. If no card is installed, then the strings that are returned will be set to whatever the current values are of the ProDOS date and time locations on the global page (00/00/00 and 00:00 normally).

If your system has no clock, and you wish to set the date and time manually, you can include the DATETIME function in your program (from your master disk).

Since ProDOS does not have any storage space for seconds, the TIME$ seconds field will always be "00".

DEF LPRINT statement

FORMAT **DEF LPRINT** [=] *Slot number*

DEFINITION This command is used to configure the printer slot during runtime. After this command is used, all printer output will be diverted to the selected slot.

The slot number may be specified by any numeric expression but the value of *Slot number* MUST be between one and seven

EXAMPLE

```
DEF LPRINT = 1
```

REMARK This command supersedes the configuration value, except for the initialization string. See the notes on configuration for more info.

If value exceeds the range of 1-7, the number will be masked to stay in range.

APPLE ProDOS APPENDIX

DEF MOUSE statement

FORMAT **DEF MOUSE** [=] *expression*

DEFINITION This statement defines which device (MOUSE or JOYSTICK) will be used for the MOUSE function call.

expression=0	APPLE MOUSE INTERFACE
expression<>0	JOYSTICK

EXAMPLE

```
DEF MOUSE=1: REM Define as a JOYSTICK
DO
  PRINT MOUSE(1), MOUSE(2)
UNTIL MOUSE(3)
END
```

This program will print the positions of the joystick until you press the joystick button.

REMARK The default is equivalent to DEF MOUSE=0. The Apple //c has the equivalent of a mouse card built-in.

If DEF MOUSE=1 is used to activate the joystick, the function MOUSE(3) will return a value corresponding to which joystick button was pressed.

Value	Meaning
0	Neither button pressed
1	Button 0 pressed
2	Button 1 pressed
3	Both buttons pressed

APPLE ProDOS APPENDIX

DIR command

FORMAT [L]**DIR** [+] [*pathname*]
[L]**CAT** [+] [*pathname*]

DEFINITION These commands display a directory of a ProDOS volume, as explained in the reference section. DIR and CAT are interchangeable. CAT was implemented to make conversion easier for Applesoft programmers.

When the command DIR is given by itself, ZBasic will display a directory of the currently logged ProDOS pathname (see the PATH command) in the standard 40 column format.

DIR+ operates in the same way as DIR without the "+", and will produce the ProDOS standard 80-column display format (more information is shown). If in 40-column mode the output will wrap to the second line.

The optional *pathname* specifies a directory to be displayed. The pathname can be a full or partial ProDOS pathname. Full pathnames start with a slash ("/"), and specify the root volume. If a partial pathname is specified, ZBasic will append this pathname to the currently logged pathname, and display the contents of this sub-directory. Pathnames can be any legal ProDOS pathname.

The optional "L" preceding the command will direct output to the printer. There must not be a space between the "L" and the "DIR".

REMARK As you can see, the first line of the directory contains the name of the directory which the listing is produced from. A slash preceding the directory name (as in the example) signifies that this is a root (volume) directory. Sub-directory names are not preceded by the slash.

The heading line is pretty much self-explanatory, except the ENDFILE (found on a DIR+ listing). The figures in the ENDFILE column represent the total number of bytes in that file.

The TYPE column represents the ProDOS standard file types, with one exception -- ZBS. This file type is a ZBasic tokenized source file.

An asterisk (*) preceding a file name signifies that this file is locked. It can not be modified in any way from within the ZBasic system.

As with the editor "LIST" command, the directory can be temporarily halted by pressing the space bar once. Pressing the space bar again will advance the directory one line. Pressing any other key will restart the listing. Pressing CTRL-C will abort the directory listing.

To read a directory from within a running program, simply OPEN the directory file as you would any other, then read the necessary information from it. See the ProDOS Technical Reference Manual, Appendix B, for information concerning the format of directory files.

Also see the special ZBasic ProDOS command: ONLINE.

APPLE ProDOS APPENDIX

EXAMPLE

```
ZBasic Ready
DIR

/ZBASIC
 NAME              TYPE  BLOCKS  MODIFIED

 ZBASIC.SYSTEM     SYS       33  10-FEB-87 12:02
*RAM.FILLER        BIN       17  31-JAN-86 11:40
*RUNTIME.OBJ       BIN       28  29-JAN-87 15:49
*ZBASIC.HLP        TXT       57  12-OCT-86 13:18
 DISKIO.BAS        TXT        7   5-DEC-86 14:26
 GRAPH.BAS         ZBS        3   6-NOV-86 10:25
 CREATE.FN         TXT        1  20-JAN-87 11:31
 DRAW.FN           TXT        3  29-DEC-86 17:08

BLOCKS FREE:    26     BLOCKS USED:  254

ZBasic Ready


ZBasic Ready
DIR+

/ZBASIC
 NAME              TYPE  BLOCKS  MODIFIED          CREATED          ENDFILE

 ZBASIC.SYSTEM     SYS       33  10-FEB-87 12:02   10-FEB-87 12:51  16384
*RAM.FILLER        BIN       17  31-JAN-86 11:40   10-FEB-87 12:51   9384
*RUNTIME.OBJ       BIN       28  29-JAN-87 15:49   10-FEB-87 12:51   7644
*ZBASIC.HLP        TXT       57  12-OCT-86 13:18   10-FEB-87 12:51    384
 DISKIO.BAS        TXT        7   5-DEC-86 14:26   10-FEB-87 12:51    844
 GRAPH.BAS         ZBS        3   6-NOV-86 10:25   10-FEB-87 12:51   1982
 CREATE.FN         TXT        1  20-JAN-87 11:31   10-FEB-87 12:51    123
 DRAW.FN           TXT        3  29-DEC-86 17:08   10-FEB-87 12:51    456

BLOCKS FREE:    26     BLOCKS USED:  254     TOTAL BLOCKS:    280

ZBasic Ready
```

NOTE: endfile numbers may not be actual.

EDITOR command

FORMAT **EDITOR** [+]

DEFINITION This command is used to enter the full screen text editor from the Standard line editor.

Typing "**EDITOR**" on the ZBasic command line will transform any program currently in memory from ZBasic tokenized format to full ASCII format and enter the full screen editor.

If you use the optional "+", the program currently in memory will have the line numbers stripped prior to entering the full screen editor. Be sure that you have not used any line number references in your program (such as GOTO or GOSUB). Use label references instead.

REMARK See the section entitled "*Full Screen Editor*" in this appendix for a complete description of editor commands and operation.

Note: To get back to the standard line editor press ESC (or CTRL-K CTRL-Q on][+).

APPLE ProDOS APPENDIX

INSLOT statement

FORMAT **INSLOT(*Slot Number*)**

DEFINITION This statement will allow you to specify the slot number of an interface card which your program is to receive input from.

This is supplied so that you can use non-standard interface cards (i.e. other than those supported directly by ZBasic, such as a graphics tablet).

Do not use this command to access a Super Serial Card; use the OPEN"C" command instead.

INSLOT(0) will "re-attach" the keyboard for input .

EXAMPLE

```
CLS
DO
  INPUT"Which slot is the widget?";Slot
UNTIL (Slot>0) AND (Slot<8)
:
INSLOT(Slot)
:
do something with the slot here...
:
INSLOT(0)          <--- Set the slot back to normal.
END
```

(example only do not use)

REMARK See your hardware technical reference manuals for details.

Note: Any interface card that attempts to store a value into an Applesoft variable (such as some clock cards) will not work correctly with ZBasic since there are no "Applesoft variables" in ZBasic.

Check the technical reference manual of the device to set it to some other parameters.

MEM command

FORMAT **MEM**

DEFINITION This command is used to show the amount of memory remaining for text and object code remaining and the amount of text and object code space used during each phase of operation.

#####	**Text**	• Shows amount of text space used
#####	**Text Mem**	• Amount of text room remaining
#####	**Object**	• Size of object code generated*
#####	**Variable**	• Amount of variable space used*
#####	**Code Mem**	• Object and variable space remaining*

*Values returned only correct immediately after compiling (RUN).

REMARK See Memory map in this appendix for the 64K or 128K ProDOS versions of ZBasic. Also see MEM in the main reference section.

APPLE ProDOS APPENDIX

MODE statement

FORMAT **MODE** [=]*expression*

DEFINITION ZBasic uses MODE to define the characteristics of a screen. ZBasic allows a program to integrate text and graphics anywhere on the screen in MODE 5 and 7. This feature allows ZBasic programs from an IBM PC and other computers to run on your Apple.

ProDOS™ Version
MODE CHART

MODE	TEXT	GRAPHICS
0,8	40 x 24	Character
1	None	40 x 48
2,10	80 x 24	Character
3	None	80 x 48
4,12	40 x 24	Character
5	40 x 24	280 x 192
6,14	80 x 24	Character
7*	80 x 24	560 x 192
9	40 x 4	40 x 40
11	80 x 4	80 x 40
13	40 x 4	280 x 160
15*	80 x 4	560 x 160

REMARK

character	= Graphics are defined as text characters
40 x 48	= Low Resolution Color Graphics
80 x 48	= Medium Resolution Color Graphics
280 x 192	= High Resolution Color Graphics
*560 x 192	= Double High Resolution. For IIe, IIc and //GS only.

Modes 9, 11, 13 and 15 have graphics at the top of the screen and text at the bottom, similar to Applesoft GR and HGR commands.

MODE will set COLOR to default, white in most modes. See COLOR for the other colors available in each mode.

*Double Hi-Res does not function in the 64K version (it requires 128k).

ON LINE command

FORMAT **ONLINE**

DEFINITION When the ONLINE command is issued in the Standard Line Editor, a list of all ProDOS volumes currently connected to the system will be displayed on the screen.

Each entry will display the slot, drive, and volume name of the device.

EXAMPLE

```
ZBasic Ready
ONLINE

S6,D1 = /ZBASIC
S3,D2 = /RAM
```

REMARK Since ZBasic will only operate on volumes by using pathnames, this is supplied so that you can identify a particular volume in a drive.

OUTSLOT statement

FORMAT **OUTSLOT(***slot number***)**

DEFINITION This command allows you to redirect output to the interface card located in the slot number specified. This is not intended as an alternative to the existing ZBasic commands (such as LPRINT or ROUTE).

This command is supplied only to allow you to interface your program with those cards that ZBasic does not support directly (such as graphics tablets, etc.).

OUTSLOT(0) will "re-attach" the screen for output.

EXAMPLE

```
CLS
DO
  INPUT"Which slot is the widget?";Slot
UNTIL (Slot>0) AND(Slot<8)
:
OUTSLOT(Slot)
:
do something with the slot here...
:
OUTSLOT(0)          <--- Set the output back to normal.
END
```

(example only do not use)

REMARK See INSLOT statement and your hardware technical reference manuals for details about using slots.

PATH command

FORMAT

PATH [[-] [-] ...] [*pathname*]
PREFIX [[-] [-] ...] [*pathname*]

DEFINITION

The PATH command allows you to set and/or display the currently logged ProDOS pathname. PREFIX is also provided for compatibility reasons.

PATH without any parameters will display the current ProDOS prefix.

"PATH *pathname*" will set the current ProDOS prefix, then display it as a verification that it was indeed set. *Pathname* can be either a full or partial pathname. If it starts with a slash ("/"), ZBasic will treat it as a full pathname and reset the prefix appropriately. If you specify a partial pathname, ZBasic will append it to the current prefix to create the new prefix.

The "-" parameter will "step-back" the current prefix by one directory. If a pathname is specified after the "-", ZBasic will then set this as the current prefix. You may use more than one "-" parameter to specify stepping back multiple directories.

EXAMPLE

```
ZBasic Ready
PATH                          (display current prefix)
/PROFILE

ZBasic Ready
PATH  ZBASIC/SOURCE           (append ZBASIC/SOURCE )
/PROFILE/ZBASIC/SOURCE        (to current prefix)

ZBasic Ready
PATH-OBJECT                   (remove SOURCE and)
/PROFILE/ZBASIC/OBJECT        (append OBJECT)

ZBasic Ready
```

REMARK

ZBasic will not allow you to remove the prefix entirely. The system ***MUST*** have a prefix set at all times.

APPLE ProDOS APPENDIX

POINT function

FORMAT **POINT** (*expression1* , *expression2*)

DEFINITION This function will return either a 0, signifying that the pixel is "off," or a 1 signifying that the pixel is "on."

EXAMPLE

```
PLOT 0,0
PRINT POINT(0,0)
:
CLS
PRINT POINT(0,0)
:
END

RUN

1
0
```

REMARK In modes 5 and 7, the POINT function cannot return the color of the pixel at the specified coordinates, due to the method that the Apple uses to create colors on the screen.

RENAME command

FORMAT **RENAME** ["] *pathname1* ["] , ["] *pathname2* ["]

DEFINITION Renames the file specified by *pathname1* to the name specified by *pathname2*. The comma separating the names IS required.

This command is supplied as an editor command in addition to the ZBasic statement so that the compiler does not have to be accessed every time you wish to rename a file.

EXAMPLE

```
RENAME ZBASIC.SYSTEM, ZBASIC
```

REMARK See RENAME in the main reference section for more information about using RENAME.

APPLE ProDOS APPENDIX

RUN* command

FORMAT See the main reference manual for syntax.

REMARK When saving your compiled programs to disk with the RUN* command, ZBasic will create a SYS type file that can be executed directly from ProDOS.

64K VERSION
In addition to your object file, the file "RUNTIME.OBJ" ***MUST*** be in the same directory.

128K VERSION
In addition to your object file, the following three files must be in the directory:

```
RT.MAIN.OBJ1
RT.AUX.OBJ0
RT.AUX.OBJ1
```

As part of it's initialization, your program attempts to load the runtime modules from disk (you don't have to do this; the compiler will generate the necessary code automatically).

If the required runtime files cannot be found, a ProDOS error message will be generated, and you will be left in the Apple system monitor.

USR function

FORMAT See the main reference manual.

REMARK When your USR subroutine is entered, the value that was in the parentheses in the ZBasic program can be found at zero page locations $64 and $65. This value will be a 16-bit integer in standard least-significant-byte/most-significant-byte order.

If your subroutine is to pass a 16-bit value back to the ZBasic program, it should place the value in locations $64 and $65, again in lsb/msb order.

128K Note: Be aware that with this version the USR routine must be located in the program auxiliary bank of memory. This is most easily accomplished by using"

```
DEF USRx=LINE nnnn
```

USR5 function

FORMAT *variable* = **USR5(*slot*)**

DEFINITION This pre-defined USR function returns the status byte of an Apple Super Serial Card in slot number *slot*.

The status byte will be returned in the lower 8 bits of *variable*.

The *variable* must be an integer variable.

If no Super Serial Card is installed in the system, the value returned will be undefined.

REMARK See OPEN"C" for more details about using communication functions and the Super Serial Card reference manual for the format of the status byte.

APPLE ProDOS APPENDIX

FULL SCREEN EDITOR

This version includes an easy-to-use, full screen text editor. It can be used to enter and edit ZBasic source program files, or any other text file. Some of it's features include full screen cursor movement, long distance cursor movement, split screen operation, cut/copy/paste/replace lines, global search, automatic indentation, full scrolling capabilities up/down and left/right, and some other goodies.

DIFFERENCE BETWEEN THE FULL SCREEN EDITOR AND STANDARD LINE EDITOR

ZBasic comes with a Standard line editor, as described in the main reference section, that works the same way on all versions of ZBasic. From this editor you can also do direct commands as described in the main reference section. You cannot do direct commands from the Full Screen Editor (other than those defined).

INVOKING THE FULL SCREEN EDITOR

To enter the full screen editor, type **"EDITOR"** from the Standard line editor ("EDITOR+" if you want to strip line numbers). If you currently have a file in memory, the file will be converted to a text file and transferred. If no file is in memory, you will enter the full screen editor without text.

RETURNING TO THE STANDARD LINE EDITOR

To return to the Standard line editor (command environment), press <ESC> (CTRL-K D in the 40-column editor). The file that you were editing will be re-loaded into the line editor (with line numbers added if the file did not contain any).

APPLE ProDOS APPENDIX

80-COLUMN EDITOR

If you are using an Apple //e with an 80-column text display, most of the functions are accessed by pressing one of the Ó or keys in combination with one of the numeric keys.

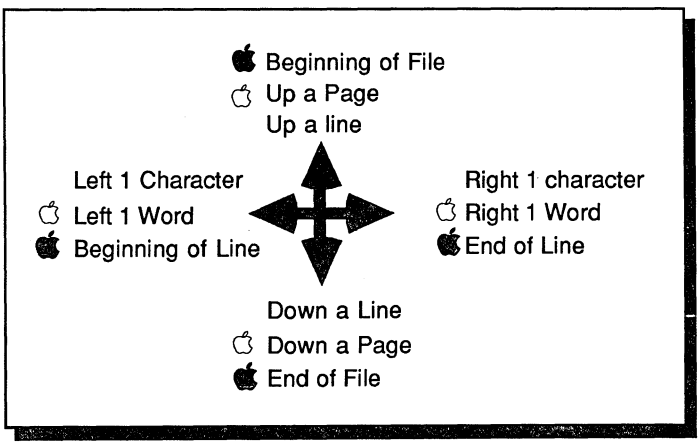
While the editor is waiting for you to enter a character, you have the option of using one of the commands available.

HELP LINE

When you press one of the Apple keys, a short "help" line will appear on the bottom line in place of the status line. This help line will remain on the screen for as long as you keep pressing an or Ó key.

The help lines are not meant to be complete descriptions of the commands available, just memory joggers.

80 COLUMN CURSOR MOVEMENT KEYS



APPLE ProDOS APPENDIX

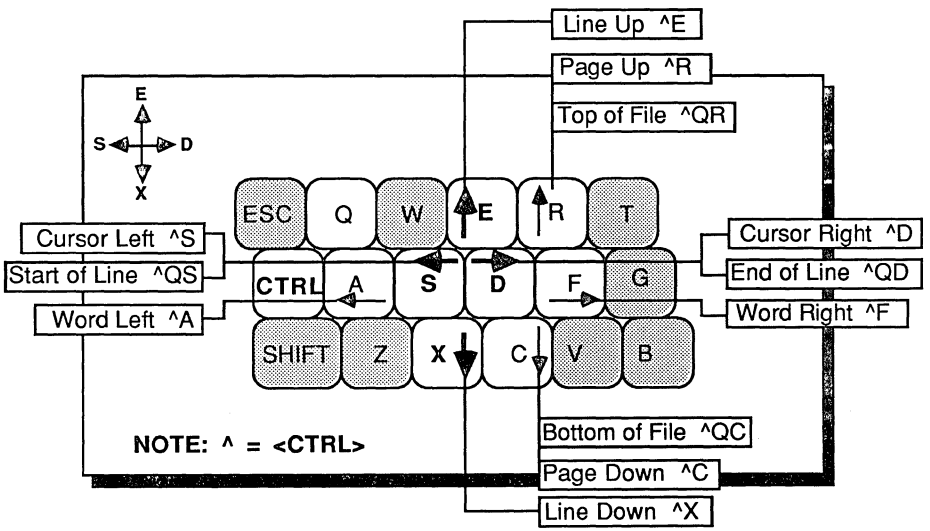
40-COLUMN EDITOR

Since Apple][+ users don't have Apple keys on their keyboard, control keys are used in place of the Apple keys. These commands have been set up to match Wordstar™, a word processor from MicroPro where possible. For those commands that are not a part of Wordstar™, we tried to make the command key match the command as logically as possible (the command is followed by an asterisk if it is not WordStar compatible).

When one of the prefix keys (Ctrl-Q or Ctrl-K) is pressed a "^Q" or "^K" appears in the lower left corner of the screen, to remind you that one of the prefix keys has been pressed. If you change your mind, and don't want to access one of the commands, simply press the space bar to cancel the command.

The following pages describe all of the commands and cursor movements available. Each one operates exactly the same way, whether the machine is a][+ or one of the newer machines.

40 COLUMN CURSOR MOVEMENT KEYS



USING THE FULL SCREEN EDITORS

The following pages contain a complete description of the Full Screen Editor. You may want to Xerox the Quick Reference page.

APPLE ProDOS APPENDIX

FULL SCREEN EDITOR QUICK REFERENCE PAGE

40 Column	CURSOR	80 Column
CTRL-S	Left 1 Character	←
CTRL-D	Right 1 Character	→
CTRL-A	Left 1 Word	Ó ←
CTRL-F	Right 1 Word	Ó →
CTRL-Q CTRL-S	To Beginning of Line	⌘ ←
CTRL-Q CTRL-D	To End of Line	⌘ →
CTRL-E	Up a Line	↑
CTRL-X	Down a Line	↓
CTRL-R	Up a Page	Ó ↑
CTRL-C	Down a Page	Ó ↓
CTRL-Q CTRL-R	To Beginning of File	⌘ ↑
CTRL-Q CTRL-C	To End of File	⌘ ↓
40 Column	**COMMANDS**	**80 Column**
CTRL-K CTRL-Q	Return to Line Editor	ESC
←	Delete Character	Delete
CTRL-Q ←	Delete to Line Start	Ó Delete
CTRL-Q CTRL-Y	Delete End of Line	⌘ Delete
CTRL-V	Switch INSERT/Overwrite	Ó -0
CTRL-K CTRL-N	Clear Text Buffer NEW	⌘ -0
CTRL-K CTRL-L	LOAD File	Ó -1
CTRL-K CTRL-S	SAVE File	⌘ -1
CTRL-K CTRL-X	Cut Line	Ó -2
CTRL-K CTRL-V	Paste Line	⌘ -2
CTRL-K CTRL-C	Copy Line	Ó -3
CTRL-K CTRL-R	Replace Line	⌘ -3
CTRL-N	Insert Line	Ó -4
CTRL-Y	Delete Line	⌘ -4
CTRL-Q CTRL-F	Find	Ó -5
CTRL-L	Find Next Occurence	⌘ -5
CTRL-K CTRL-I	Set Tab Value	Ó -6
CTRL-Q CTRL-I	Autotab On/Off	⌘ -6
CTRL-K CTRL-F	Restore Line	Ó -7
CTRL-K CTRL-P	LLIST	⌘ -7
CTRL-Z	Scroll Up	Ó -8
CTRL-W	Scroll Down	⌘ -8
CTRL-K CTRL-T	Freeze Top	Ó -9
CTRL-K CTRL-B	Freeze Bottom	⌘ -9

CURSOR KEY DEFINITIONS

This page contains the detailed descriptions of the cursor key movements for the Full screen editor:

80 COLUMN 40 COLUMN

UP A LINE

Up-Arrow **CTRL-E**
Moves the cursor up one line.

UP A PAGE

Ó-Up-Arrow **CTRL-R**
Moves the cursor one page back in the file. A page is defined as the current number of lines in the editing window minus one.

UP TO TOP

Ô-Up-Arrow **CTRL-Q CTRL-R**
Places the cursor at the beginning of the file.

LEFT A CHARACTER

Left Arrow **CTRL-S**
Moves the cursor one character to the left.

LEFT A WORD

Ó-Left Arrow **CTRL-A**
Moves the cursor to the beginning of the word to the left of the current cursor position.

LEFT TO START OF LINE

Ô-Left-Arrow **CTRL-Q CTRL-S**
Moves the cursor to beginning of current line.

RIGHT A CHARACTER

Right Arrow **CTRL-D**
Moves the cursor one character to the right.

RIGHT A WORD

Ó-Right-Arrow **CTRL-F**
Moves the cursor to the beginning of the next word to the right of the present position.

RIGHT TO END OF LINE

Ô-Right-Arrow **CTRL-Q CTRL-D**
Moves the cursor to the end of the current line.

DOWN A LINE

Down Arrow **CTRL-X**
Moves the cursor down one line.

DOWN A PAGE

Ó-Down-Arrow **CTRL-C**
Moves the cursor down one page in the file.

DOWN TO END OF FILE

Ô-Down-Arrow **CTRL-Q CTRL-C**
Moves the cursor to the end of the file.

APPLE ProDOS APPENDIX

FULL SCREEN EDITOR COMMANDS

This following pages contain the definitions for the full screen editor commands (cursor movement definitions are on the previous page).

80 COLUMN **40 COLUMN**

DELETE CHARACTER

DELETE **Left Arrow***

Deletes the character to the left of the cursor. If the cursor is currently at the beginning of the line, then the editor will assume that the user means to delete the carriage return at the end of the previous line. The current line and the previous line will be combined, and the cursor will be placed at the old end of the previous line.

DELETE TO BEGINNING OF LINE

-DELETE **CTRL-Q Left Arrow***

Deletes characters from the beginning of the line through the character to the left of the cursor. The remainder of the line will be moved to the left.

DELETE TO END OF LINE

-DELETE **CTRL-Q CTRL-Y**

Deletes characters from the current cursor position to the end of the line.

QUIT THE FULL SCREEN EDITOR

ESC **CTRL-K CTRL-D**
CTRL-K CTRL-Q

This command quits the Full Screen Editor and returns to the Standard line editor. Any text that is currently in the text buffer will be re-loaded into the line editor, with line numbers added to each line if the text does not already contain line numbers.

INSERT / OVERWRITE

-0 **CTRL-V**

This command is another toggle, switching the editor between Insert and Overwrite modes of operation. The editor "wakes-up" with overwrite mode selected (as can be seen on the bottom status line). The overwrite cursor is the underline character. While overwrite mode is active, any characters that you type will replace whatever character the cursor is currently on (except for the carriage return character at the end of a line). If the cursor is at the end of a line, then any characters that you type will effectively be inserted ahead of the terminating carriage return.

When Insert mode is selected, the cursor character changes to the caret ("^") character, and any characters that you type will be inserted at the current cursor position, moving any characters at and to the right of the cursor over one position to the right. If you press RETURN while in the middle of a line, the cursor will be moved down a line and to the left margin, and the portion of the line at and to the right of the cursor will be brought down as well.

The current setting of the Insert/Overwrite switch can be seen on the status line.

NEW (Clear Text Buffer)

-0 **CTRL-K CTRL-N***

This command will clear any text from the text buffer and set the search string to null. It will then clear the active window, and place the cursor in the upper left corner of the window. It also removes the current file name from memory, and selects overwrite mode. If no file is in memory when the editor is entered, this is the state that is set when the editor "wakes-up."

LOAD A FILE

-1 **CTRL-K CTRL-L***

This commands clears any text from the text buffer, and then will prompt you for the ProDOS pathname of a file to load. The file MUST be a TEXT type file (TXT). If it isn't, you will receive an error message.

If you have previously loaded a file, the system will place the file name of this file on the screen for you automatically. If this is the file that you wish to re-load (if, for example, you really botched up the file and want to start over again), simply press the return key. If you want a different file altogether, press CTRL-X to remove the old file name, and enter the new file name. The left arrow key or the delete key can be used to correct any typing errors. If you initiate this command by accident, you can press CTRL-C to return to the editor with your current file intact.

SAVE A FILE

-1 **CTRL-K CTRL-S**

This command will prompt you for a ProDOS pathname to save the current text. If you have previously used the Load command to load a file into the buffer, the system will place the current file name on the prompt line for you. You have the same options here as you did when you loaded the file. Use caution with this command. If a file already exists on the disk with the same file name, the editor will replace whatever was previously in the file without any warning message.

CUT LINE

-2 **CTRL-K CTRL-X***

This command will remove the current line from the text buffer and place it on the clipboard (just a temporary holding area). The line will remain on the clipboard until you either Cut another line, or Copy a line. This line can be pasted from the clipboard back into the main text buffer with the Paste command. This command will only work with entire lines. There is no way to Cut in increments of less than, or more than, a single line.

(This command actually does a COPY, and then a DELETE.)

PASTE LINE

Ó-2 **CTRL-K CTRL-V***

This command will copy whatever line is currently on the clipboard into the current position within the main text buffer. The current line will be moved up in the buffer (down on the screen) to make room for the new line. This action does NOT remove the line from the clipboard. Therefore, you can paste the same line as often as you like, into as many places in the text as you like.

COPY LINE

Ó-3 **CTRL-K CTRL-C***

This command will make a copy of the current line to the clipboard. It does not remove the line from the main buffer. You are then free to paste this line to your liking.

REPLACE LINE

Ó-3 **CTRL-K CTRL-R***

This command will replace the current line with the line that is currently on the clipboard. If there is no line currently on the clipboard, no action will be taken.

INSERT LINE

Ó-4 **CTRL-N**

This command will insert a carriage return at the current cursor position without moving the cursor. If the cursor is at the beginning of a line, then the current line will move down on the screen, leaving a blank line for you to enter a new line on. If the cursor is in the middle of a line, the portion of the line at and to the right of the cursor will be moved down to the next line, and the cursor will remain at the (new) end of the current line.

DELETE LINE

Ó-4 **CTRL-Y**

This command deletes the current line from the text. No copy of the line is retained in memory. Therefore, this is not a reversible command. Use it with caution.

FIND

Ó-5 **CTRL-Q CTRL-F**

This command will allow you to enter a character sequence of up to 30 characters. The editor will then search from the current position to the end of the file for the character sequence. If it can't find the search string, a message will be displayed on the last line to this effect, and the cursor position will not change. If the string is found, the line containing the string will be placed at the current cursor position, and the cursor will be placed at the beginning of the string.

FIND NEXT OCCURRENCE

Ó-5 **CTRL-L**

This command searches for the last search string that was entered using the FIND command. This command operates in exactly the same way as the find command, except that it does not prompt for the search string. (As a matter of fact, the find command prompts for the search string, and falls through to this command.)

SET TAB STOP

Ó-6 **CTRL-K CTRL-I**

This command will allow you set the size of the tab stops. The editor will prompt you for the new value on the bottom line of the screen. The last part of the prompt is the current value of the tab setting, which has a default of 16. To leave this setting, simply press the RETURN key. To change it, enter the new value. The editor uses the same tab value as the rest of the ZBasic system, so this command accomplishes the same thing as the DEF TAB statement in a ZBasic program. This also implies that if the value is changed here, then the value will be changed for the rest of the system as well.

This tab value is used in the screen editor whenever you press the TAB key (CTRL-I for][+ users). The cursor will be positioned to the next calculated tab stop on the screen. If the next tab stop is beyond the end of the line, the cursor will be placed AT the end of the line. If you continue to press the TAB key, the cursor will move to the beginning of the next line (the absolute first tab position on the screen), and then continue normally.

AUTO TAB ON/OFF

Ó-6 **CTRL-Q CTRL-I***

This command is simply a toggle to turn the autotab feature on or off. The editor starts with autotab on. The current setting of autotab can be seen on the status line at the bottom of the screen. Autotab is a feature that will allow you to enter nicely formatted source code. When Insert mode is on, and RETURN is pressed at the end of a line, the cursor will be moved down to the next line, and then spaces will be inserted in the new line until the cursor is in a position underneath the first non-space character in the line above. If Insert is off (Overwrite mode), autotabbing is only operable when you are entering text at the end of the file. This is so that spaces are not inserted ahead of any existing text.

RESTORE LINE

Ó-7 **CTRL-K CTRL-F***

This command will restore the line at the current cursor position, deleting any changes that you have made to the line. This will only work if the cursor has NOT moved off the line since you made the changes, and/or the screen has not scrolled sideways. Normally, while you are editing a line, you are actually editing a copy of the line in an edit buffer. When the cursor is moved off the line, or if the screen is scrolled either left or right, this edit copy of the line is moved back to the main text buffer prior to moving the cursor. If you have made some changes to a line, and then change your mind, an old copy of the line still resides in the main buffer. A

copy of this old line is placed into the edit buffer over any changes that you might have made when you invoke this command.

PRINT LINE

Apple-7 **CTRL-K CTRL-P**

This command will print the contents of the text buffer to your printer. This command accomplishes the same operation as the line editor's LLIST command without any parameters. The entire contents of the text buffer will be printed; no provision is provided for printing only a portion of the buffer. If no printer is connected in the slot that is currently configured, the system may "hang". Press CTRL-RESET to warm start the editor.

SCROLL DOWN

Open-Apple-8 **CTRL-W**

This command will scroll the screen down, with the cursor remaining in the same screen position (which means that it will be on the previous text line). If the cursor is currently within the first page of the text, the screen will not scroll, but the cursor will move up a line. The cursor will ***NOT*** move past the first line of the file.

SCROLL UP

Apple-8 **CTRL-Z**

This command will scroll the screen up, with the cursor remaining in the same screen position (which means that it will be on the next text line). If the cursor is currently within the last page of the text, the screen will not scroll, but the cursor will be moved down a line. The cursor will ***NOT*** move past the last line of the file.

FREEZE SCREEN FROM THAT LINE UP

Open-Apple-9 **CTRL-K CTRL-T***

This command will freeze the top of the screen. A separating line will be drawn at the current cursor position, and the portion of the screen above this line will be frozen. The line that the cursor is in will remain within the new active portion of the screen (the active window). While the screen is frozen, the cursor will not move into the frozen portion, and no changes will be made within the frozen portion.

Only the screen is frozen. You can still move the lines that are in the frozen portion into the active window and make changes, but these changes will not appear in the frozen window.

To "thaw out" the window, press the control key sequence again. The separating line will be removed, and the screen will be refreshed, leaving the cursor at whatever position it was at.

FREEZE SCREEN FROM LINE DOWN

Apple-9 **CTRL-K CTRL-B***

This command freezes the bottom portion of the screen. A separating line will be drawn at the current cursor position, and the portion of the screen below this line will be frozen. The text line that the cursor was on will remain within the active window. This command is very much like the Freeze Top command. To "thaw out" the bottom window, press the command key sequence again.

These two Freeze commands are entirely separate from each other. You can have up to two frozen windows on the screen: a top window and a bottom window, leaving a third, active window in the middle of the screen between the two frozen portions.

This can be handy if you have a couple of different subroutines in a couple of different sections of your program, while accessing those subroutines in a third portion of your program.

For Macintosh Programmers...

CONSTRUCTION SET

A utility that brings you a MacDraw-like environment for easily creating:

WINDOWS
EDIT FIELDS
MENUS
BUTTONS
SCROLL BARS

Very little typing.... just draw with the mouse. You can set up complicated EDIT FIELD, BUTTON, MENU and WINDOW layouts in minutes (that used to take many incredibly boring, tedious, hours).

The ZBasic Construction Set creates a ZBasic Source file that you just merge into your programs. Created by a ZBasic programmer that got tired of spending a couple of days for each program *JUST TO DO THE SCREEN LAYOUTS!*

Best of all the price is right:

only

$49.95

79.95

plus shipping and handling: $5 U.S., $12 Canada, $25 Overseas

The ZBasic Construction Set is available now only from Zedcor direct:

800-482-4567

ZEDCOR

4500 E. Speedway, #22
Tucson, Arizona 85712
(602) 881-8101

MACINTOSH APPENDIX

ZBasic Macintosh™
Version 4.0

For the Macintosh 512K, Macintosh Plus,
Macintosh SE, Mac XL and LISA® with MacWorks™
and the Macintosh II

Additions and Enhancements to 4.0
by
Andrew Gariepy

Original version by
Andrew Gariepy, Dave Overton and Scott Terry

MACINTOSH APPENDIX

TABLE OF CONTENTS

MACINTOSH APPENDIX

MACINTOSH APPENDIX

MACINTOSH APPENDIX

GETTING STARTED

Welcome to ZBasic, the most powerful and easy to use language available for the Macintosh. Follow these instructions and you'll be up and running in no time.

◆ MAKE A BACKUP FIRST

ZBasic is not copy protected. Make a copy or two for your personal backup purposes. Put the master diskette in a safe place for future use. It's a good idea to set the write protect tab on the master diske.

◆ MOVE MAIN PROGRAMS TO AN HFS SYSTEM DISK

ZBasic is provided on a 400K single sided diskette. The diskette is readable by any Macintosh. It doesn't come with a system so you'll need to copy files to a system diskette or hard disk. The main files to copy are: ZBasic™ and ZBasic.HLP. If you plan on using MacinTalk copy this file to your system folder. All other files are example programs and may be copied at your discretion.

◆ INSTALL PRINTER DRIVERS

If you haven't done so already , be sure to put a printer driver in your new system folder.

◆ USE FONT/DA MOVER TO INSTALL THE ZBASIC DESK ACCESSORIES

Use Font/DA Mover to move the ZBasic MEMORY MONITOR and TRON MONITOR Desk Accessories over to your system. These DA's are very useful for debugging purposes (see MEMORY MONITOR and TRON MONITOR in the reference section for explanations and examples of use).

◆ NOTE DIFFERENCE BETWEEN THE "COMMAND WINDOW" AND "EDIT WINDOW"

The EDIT Window is the Macintosh-type editor and will probably be the one you use the most. The Command Window is the "Standard Line Editor" and is the one being referred to in the front of this manual. Use <COMMAND E> to switch from one to the other. Be sure you understand the difference between these two modes before proceeding.

◆ READ "GETTING STARTED" IN THE FRONT OF THIS MANUAL

Read "Getting Started" in the front of this manual to get the feel of ZBasic.

◆ READ THIS APPENDIX

Read through this appendix and get an idea of the Macintosh specific information that is available like; the Mac specific commands in this reference section, using the Toolbox, Converting MSBASIC programs, "How to write a Macintosh Application", and so on.

◆ TRY THE EXAMPLE PROGRAMS

Try out some of the example programs we've provided on the master disk. Of special interest is the Thinner.BAS program by Andrew Gariepy. It illustrates how easy it is to create Macintosh-type applications using ZBasic. There are also example programs in the toolbox section you may want to type in and try.

◆ CREATE YOUR OWN PROGRAMS

Now you can start creating your own programs. Still got questions?
Call us at (602) 795-3996 and we'll be glad to help you.

MACINTOSH APPENDIX

FILES INCLUDED ON THE MASTER DISKETTE

	The ZBasic™ BASIC Compiler (version sometimes shown).
ZBasic™.HLP	The Help File. Select from Apple menu or type HELP n from Command window.
Macintalk™	Macintalk file licensed from Apple. Must be in the system folder of an application's start-up disk.
MEMORY Monitor DA	ZBasic desk accessory. Described in the reference section.
TRON Monitor DA	ZBasic desk accessory. Described in the reference section.

ZEXAMPLE FOLDER

HOUSE	Examples of "Standard graphics".
PYRAMID	Examples of "Standard graphics".
THINNERxxx.BAS	A MacPaint type program that will allow you to load, modify and save out Macpaint files and Digitized TIFF files. Great example of doing Macintosh graphics (xxx is the version number).
SCIFN.BAS	A number of Scientific Math FN's you can use in your programs.
SORT.BAS	Example program to be used with QUICK.APP and SHELL.APP
QUICK.APP	Two sort routines you can customize for your applications. QUICK is
SHELL.APP	good for a large sort. SHELL is good for smaller sorts.
IOTEST.BAS	Example of using Windows, Edit fields and more.
NEWFILE.BAS	Get pathnames at runtime.
GETSCRAP.BAS	Example of getting things off the clipboard.
GETPUT.BAS	Example of doing Window refresh with GET/PUT statements.
WINDOWPIC.BAS	Example of Window refreshing with WINDOW PICTURE statement.
PICT.BAS	Example of deleting a PICTURE handle from memory.
SEGEXAMPLE.BAS	Example of using SEGMENT and SEGMENT RETURN

How to Install Icons in your Applications: Numerous examples for putting icons into your applications. A Complete description of how to use these files is included in this appendix:

RMaker™:	Resource compiler licensed from Apple Computer, Inc.
IOTEST.R:	RMaker™ source file example program for installing Icons.
XICON.Rsrc	Example Icon resource.
ResEdit	Not included with ZBasic...but you may want it, or some other Icon editor, for adding or editing application icons.

Note: These were the examples as of 5/87, check the disk, there may be other examples.

MACINTOSH APPENDIX

NOTES TO THE MACINTOSH VERSION

ENHANCEMENTS TO VERSIONS BEFORE 4.0

The following items have been added to this version of ZBasic:

SELECT CASE Structure: Makes structured programming even easier.
TRON MONITOR Desk Accessory: Trace variables and program flow at runtime.
NEW-VASTLY IMPROVED-EDITOR: We've eliminated Apple's EDIT and the old EDIT Window. Note that you can now do most commands from the editor environment. Now you don't need line numbers at all!! Set for no line numbers under "Configure".
128K ROM SUPPORTED: Especially in the area of Toolbox calls and function. The biggest additions are with LIST MANAGER and SCSI calls.
NEW MANUAL: You'll find the examples and syntax improvements in the toolbox area and the alphabetized reference section especially helpful.
NEW COMMANDS like: TEHANDLE and GET WINDOW . See the reference section for details.

EXECUTING ZBASIC FROM THE FINDER

Insert a backup of the ZBasic™ diskette into either drive. Double-click the ZBasic icon.

KEYBOARD VARIATIONS IN THE COMMAND WINDOW

The ZBasic manual makes reference to certain keys for certain purposes. The actual keys you will use on the Macintosh may be different. Wherever reference is made to a key that does not exist, use this chart to find out which key to use:

Reference manual Key	Macintosh Key
<ENTER>	<RETURN>
<BREAK> or <CNTR C>	<Command period>or <OPTION C>
<ESC>	<TAB>
<up/down arrows>	<[>, <]> *
List the First/Last line in program	<{>, <}>

* Macintosh Plus™ cursor keys also work

NOTE: The keys above apply to the COMMAND WINDOW only.

ICON TYPES

When you save programs or files you will see these icons:

filename
Source code saved in Text format (ASCII): SAVE+ or SAVE*. File type of TEXT

filename
Source code saved in tokenized format (condensed): SAVE File Type of ZTKN

filename
Chain file created with RUN+ (no runtime included) File type of ZCHN

filename
ZBasic DATA file created from a ZBasic application. File type of ZDAT.

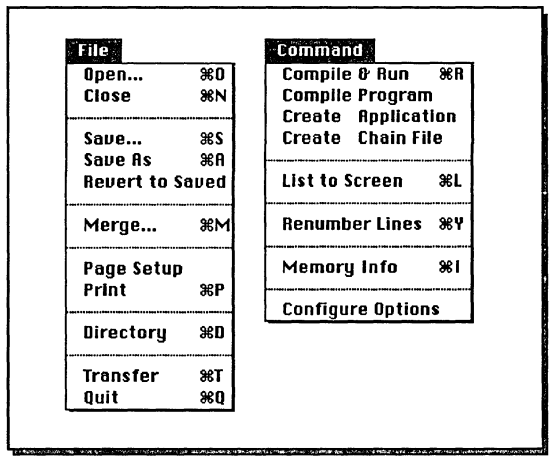
The CREATOR type of all ZBasic files is "ZBAS". See CREATOR, DIR and DEF OPEN in this appendix.

MACINTOSH APPENDIX

THE COMMAND WINDOW

The "Command Window" is the standard ZBasic™ line editor (as described in the main section of the manual).

Most direct and line editing commands should be typed in from this window. Some commands are also available under menu items:



MENU COMMANDS

Note that many of these menu items correspond directly to ZBasic commands:

File Menu	ZBasic Equivalent	Command Menu	ZBasic Equivalent
Open...	LOAD	Compile and Run	RUN
Close	NEW	Compile Program	COMPILE
Save	SAVE (in last format)	Create Application	RUN*
Save As	SAVE, SAVE+ or *	Create Chain file	RUN+
Revert	(loads last version)	List to Screen	LIST
Page Setup	Mac page set-up	Renumber lines	RENUM
Print	LLIST	Memory info	MEM
Directory	DIR	Configure options	CONFIG

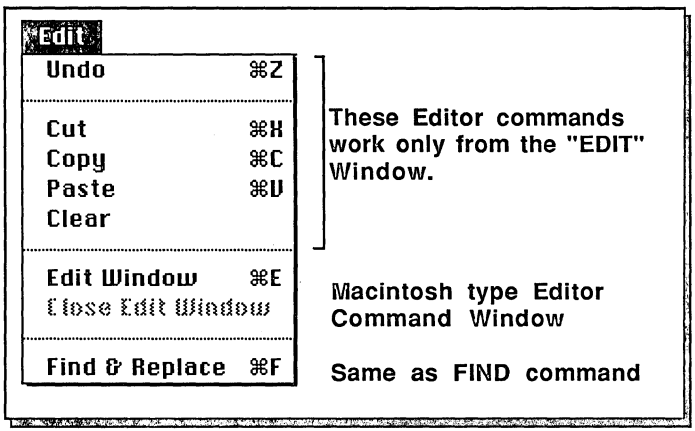
Transfer	Execute another application directly without going to the desktop
Quit	Exit and go to the Finder.

MACINTOSH APPENDIX

THE EDIT WINDOW

The EDIT window is the Mac-type editor where you'll probably be spending most of your time. Press <Command E> from the Command window or select "Edit window" from the "Edit" menu. The title bar of this window will display "Edit -- Your Filename".

The EDIT window allows you to cut, paste and copy text with almost the same commands you're use to from other Macintosh editors like MacWrite. You can even select font, style and size to suit your preferences.



NO LINE NUMBERS

Under "Configure" there is an option to "Use Line numbers to Order Text". Disable this item and line numbers will not be required when editing from the Full Screen Editor.

You will still be required to select "Save without Line numbers" the first time you save your program. After which ZBasic will automatically save it without line numbers when you select the SAVE item or press <COMMAND S>.

ERRORS

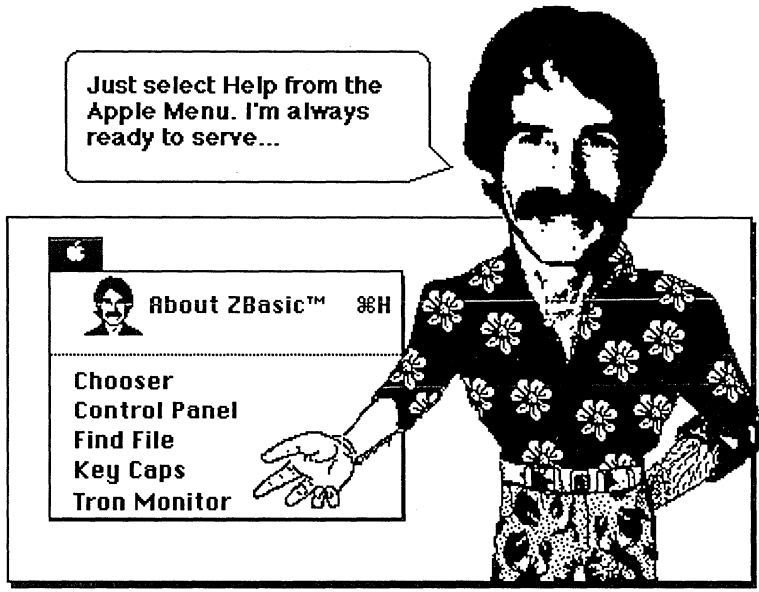
When you compile a program and an error is encountered, the error is displayed in the Command Window and the cursor of the Edit Window will be on the line where the error occurred.

For this reason it is a good idea to keep a portion of the Command Window visible when compiling. You may size and position the windows to your liking.

MACINTOSH APPENDIX

HELP WINDOW

HELP is available as a window under the menu and by typing HELP from the command window. Use the scroll bar to scroll through the window contents for the information you need. That's Andrew's icon there. (Yes... he really does wear clothes like that!)



HFS versus MFS

ZBasic works great with both MFS and HFS (MFS is the old Macintosh filing system). See FILES$ in this appendix for determining system type and more.

LOADING OLD ZBasic™ PROGRAMS

ZBasic tokens were changed slightly from older versions so you will need to do the following to convert your programs that were not saved in text (ASCII) format:

1. Load Older version of ZBasic
2. Load your old Program (Tokenized source code saved with SAVE). (A tokenized file has a desk top icon with O's and 1's in it.)
3. Save program with SAVE* (ASCII-TEXT format)

----------Program may now be loaded correctly ----------

Failure to convert tokenized programs with this procedure may result in some of the keywords being wrong when you load your old programs. This will produce unexpected syntax errors and possibly other compiler or runtime errors.

MACINTOSH APPENDIX

MEMORY AVAILABLE TO VARIABLES

VARIABLE TYPE	MEMORY AVAILABLE TO PROGRAM
Array variables	All available memory.
INDEX$ variables (10 available)	All available memory. See MEM(-1) and INDEX$
Regular (A%, A&, A$, A!, A#)	A maximum of 128K total for all of these type.

MACHINE LANGUAGE

Machine language routines and calls must preserve registers; A4, A5, A6, A7. To insure compatibility with future versions of ZBasic for 68000 systems, registers D4, D5, D6, D7 should also be preserved.

USR(0) through USR(9) functions pass the argument as a longword in register DØ. Function return parameters (function results) must be returned in DØ.

LONG INTEGER

The Macintosh version of ZBasic provides two types of integers:

TWO BYTE INTEGER (%)	FOUR BYTE LONG INTEGER (&)
-32,768 to +32,767	-2,147,483,648 to +2,147,483,647

Both types are supported only to remain compatible with storage requirements of previous versions of ZBasic. Note that "intentional" overflow errors will produce different results on this version since there is 32 bits instead of 16 bits and the sign will not change as before.

To define specific variables as Long integer (four byte integer) use DEFDBL INT at the beginning of a program. You may also use "&" as a Long integer variable indicator. Long& would signify the variable "Long" as a four byte integer. Short% would be a two byte integer variable.

FILE AND MEMORY REQUIREMENTS OF LONGINTEGERS

When Long integer variables or numbers are written to disk or stored in memory they use four bytes.

SETTING DEFAULTS TO LONG INTEGER

In other versions of ZBasic MKI$, HEX$, BIN$ and OCT$ will only function in two byte integer. This version will use LONG integer functions when you use the statement: DEFSTR LONG

To switch back to two byte WORD (16 bit) use: DEFSTR WORD. The default setting is DEFSTR WORD. Also see PEEK WORD, PEEK LONG, POKE WORD and POKE LONG.

MACINTOSH APPENDIX

FLOATING POINT ENHANCEMENTS

The floating point precision for this version of ZBasic may be configured from 8 to 240 digits of precision (we don't know of any other language that gives you this capability). New parameter rules:

Accuracy is selectable up to 240 digits
Double Precision configuration starts at 8 digits
Single Precision configuration starts at 2 digits
Scientific Precision configuration starts at 4 digits

The range of double precision has been expanded to ±E-16,384 to E+16,383. Single precision range stays the same as other versions of ZBasic with ±E-64 to ±E+64 to maintain compatibility. Default storage requirements are 8 bytes and 4 bytes for Double and Single precision respectively.

Note: For compatibility reasons, 12 digit double precision actually returns 14 digits of precision internally and stores the values as 12 digits.

10,000 DIGITS OF FLOATING POINT ACCURACY?

A Secret feature: You may configure Scientific functions up to 1024 digits, and Add, Subtract, Multiply and Divide to 10,000 digits if you use a "™" after the number used with Double Precision when configuring ZBasic (see "Configure"). This may cause some problems when printing at certain times..

FLOATING POINT SPEED

INTEGER CALCULATIONS ARE ALWAYS MUCH FASTER THAN FLOATING POINT. Whenever speed is important, use integer math instead of floating point.

If you use floating point numbers in the range of ± 2 million, with only 2 digits of fraction needed, consider using Long integers and divide by 100. Speed will be considerably faster. Totals or certain variables that would contain numbers outside this range should use single or double precision numbers, of course.

Also see USR for high speed integer SIN, COS and SQR.

HOW TO GET MAXIMUM SPEED OUT OF BCD FLOATING POINT

To get maximum speed out of floating point calculations, set the digits of precision under the "Configure" menu to:

DOUBLE PRECISION	8 digits
SINGLE PRECISION	6 digits
SCIENTIFIC PRECISION	4 digits

Note: The speed of Single and Double precision is the same, only the memory for storage on disk and RAM is different. If speed is more important than accuracy, the digits of precision set for Double precision is the deciding factor.

When more speed is important consider using Long integers with the fixed point routines in the Macintosh toolbox for a high speed floating point range of ±32,767.9999.

MACINTOSH APPENDIX

FLOATING POINT SPEED VERSUS ACCURACY

The following chart will give you an idea of the performance of floating point math with given digits of accuracy. The more speed you need the smaller the digits of accuracy will have to be set (under "Configure menu").

The benchmark programs are:

BENCHMARK#1

```
DEFDBL A-Z
S#=TIMER
FOR I%= 1 TO 1000
  X=X+ABS(EXP(LOG(I%))-SQR((I%/1.)
  ^2)+(2./3.)*(I%*3./2.)-I%)
NEXT I%
PRINT TIMER-S#,X
```

BENCHMARK#2

```
DEFDBL A-Z: X = 3.14159
S#=TIMER
FOR I%=1 TO 1000
  X=X+I%: X=X/I%
  X=X*I%: X=X-I%
NEXT I%
PRINT TIMER-S#, X-3.14159
```

Digits of accuracy set under "Configure"			Benchmark#1		Benchmark#2	
DBL	SNG	SCI	Seconds	Error (X)	Seconds	Error(X)
8	6	4	65	10.38	6	.418
8	6	4	65	10.38	6	2E-4
8	6	8	88	6.7E-3	6	.418
8	6	8	88	6.7E-3	6	2E-4
12	6	8	105	4.52E-4	7	.418
12	10	8	105	4.52E-3	7	7E-8
12	10	12	130	6.75E-4	7	7E-8
16	14	12	183	2.55E-8	8	9E-10
16	14	12	190	6.54E-11	8	6E-12
16	14	16	270	7.05E-15	8	6E-12
20	18	20	270	7.05E-15	9	1E-14
20	18	20	360	7.16E-19	9	7E-16
24	18	24	608	7.13E-27	11	6E-20
32	18	32	1327	7.32E-43	14	2E-28
48	18	48	2425	7.63E-59	20	6E-44
64	18	64			27	1E-60
96	18	96			45	4E-92
128	18	128			66	1.7E-123
192	18	192	33283	8.2E-187	110	2E-188
240	18	240			166	2E-236

DBL = Double Precision
SNG = Single Precision
SCI = Scientific Precision

Note: Benchmarks were done on a standard 512K Macintosh™.

MACINTOSH APPENDIX

FILE LENGTH ENHANCEMENTS

Unlike other versions which have are limited to 65,535 records and a record length of 65,535, this version allows up to 2 billion records (LongInteger) and each record may be up to 2 billion bytes long (hardware and memory permitting, of course).

RS-232 COMMUNICATIONS

There are a number of enhancements to the standard OPEN"C" statement. See OPEN"C" and HANDSHAKE in this appendix for details.

DETERMINING WHETHER 128K OR 64K ROMS ARE INSTALLED

You will find that there are some statements and toolbox routines that will only operate with the 128K ROM which is found primarily on the Macintosh PLUS, SE and MAC II as well as enhanced Mac 512K. To determine which ROM is installed run this example:

```
IF PEEK(&28E) AND 128 THEN PRINT "OLD ROMS" ELSE PRINT"NEW ROMS"
```

SPEED AND EVENT TRAPPING

Since ZBasic must check for an event at the beginning of each line, program execution will be significantly slower during event trapping. To optimize program performance, use event handling only when needed and do *event type* **OFF** for loops or other structures that require high performance.

Event type **STOP** does not improve program speed. ZBasic continues checking for events and stores them in the queue.

DEBUGGING PROGRAMS

This version of ZBasic offers a number of powerful debugging features including the TRON MONITOR, MEMORY MONITOR, TRON V statement and BREAK ON statement. See the reference section of this appendix for details.

BREAKING OUT OF PROGRAMS

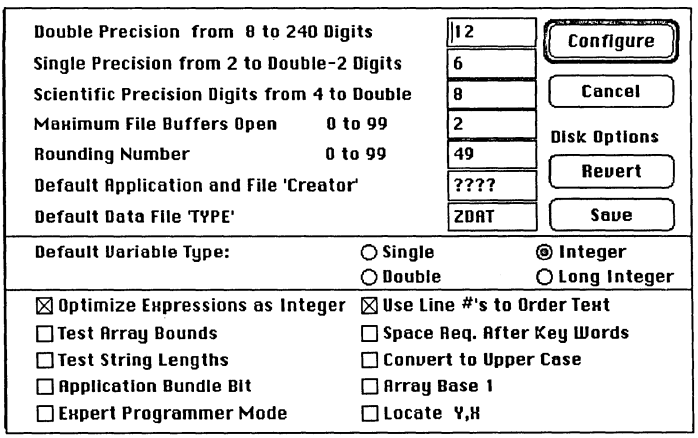
ZBasic is a compiler, therefore programs cannot be "Exited" unless you specifically tell ZBasic to do so. Use TRONB or TRONX (or BREAK ON) to enable the <Command (.) > break key (or OPTION C for DIR and LIST). This version of ZBasic adheres to the Macintosh standard of <Command period (.)> to stop or break out of a program only if TRONB or TRONX is used. You may also use <Option C> .

BREAK ON, BREAK OFF and ON BREAK may also be used from within a program to enable the <Command Period>. Note: BREAK ON *WILL SLOW EXECUTION SPEED.*

MACINTOSH APPENDIX

MACINTOSH SPECIFIC CONFIGURATION OPTIONS

When you select "Configure" under the "Command" menu you will be offered a number of options:



You will recognize most of the options being offered as the same as those described under "Configure" in the front of this manual. Special notes to those not covered:

- Double and Single precision may be configured up to 240 digits (10,000 if you use "™").
- See CREATOR and DEFOPEN for definitions of these options.
- "Save" saves your options so they are the same next time you load ZBasic.
- "Revert" sets all options back to the original default.
- "LongInteger" is now allowed as a default variable type.
- "Use Line #'s to Order Text" lets you work with or without line numbers from the "EDIT WINDOW". Note that line numbers are always added in the "Command Window" since it would not be possible to use it without them. See "SAVE AS" under File menu.
- "Space req. after Keywords" forces you to put spaces after keywords but allows you to embed keywords in your variables. This also makes programs a little easier to read.
- "Expert Programmer Mode" does things without asking "if you're sure". Not for the weak of heart.
- "LOCATE Y,X" reverses the vertical and horizontal coordinates for LOCATE.
- "Optimize Expressions as Integer": See "Expression Evaluation" in "Math section of this manual. Used primarily for compatibility with MSBASIC (set to NO to do that).

MACINTOSH APPENDIX

HOW TO CREATE MACINTOSH APPLICATIONS

The Macintosh version of ZBasic is a different animal than versions running on other computers. Sure, programs from other versions will run on the Mac, but unless you add the "Macintosh touch" to these programs your users will not be be happy.

THE MACINTOSH INTERFACE

ZBasic provides access to almost every part of the Macintosh Read Only Memory (ROM), both indirectly through the many "Mac" type statements and functions built into ZBasic and directly through the use of the Mac toolbox.

To make your programs understandable to Macintosh users you must adhere to a number of standards and understand the following terms:

--------- TERMS ---------	---------- WHERE TO LOOK ----------
• MENUS	• MENU, APPLE MENU, MENU ON/OFF
• WINDOWS	• See WINDOW, WINDOW PICTURE GET WINDOW, DIALOG
• SCROLL BARS, BUTTONS	• BUTTON, SCROLL BUTTON, DIALOG
• TEXT, FONT, STYLE, SIZE	• See TEXT in this reference and toolbox.
• MAC STYLE TEXT EDITING	• EDIT FIELD, EDIT$, EDIT MENU, TEHANDLE, TE section of Toolbox, SCROLL BUTTON example program shows scrolling of text files in a window.
• MOUSE	• MOUSE, MOUSE ON/OFF, CURSOR, DEFMOUSE (also MOUSE in main ref.)
• SAVE/LOAD FILES	• FILES$, OPEN, APPEND
• GRAPHICS	• GET/PUT, PICTURE (PICT) and the QUICKDRAW section of the Toolbox, loading and saving "MacPaint files" in this section and Tif and PostScript.
• RESOURCES	• See PICTURE, Resource Manager in Toolbox, OPEN"RO", "RI", RR", "RA".
• ICONS	• See ICON example in toolbox, and example of creating an icon for your application in this appendix.
• PRINTING	• See section in this appendix "Printing" and LPRINT, ROUTE 128, DEF LPRINT DEF PAGE and CLEAR LPRINT.
• CLIPBOARD	• See CLIPBOARD in this section.

THINGS TO AVOID

You should normally avoid non-Macintosh type commands like:

AVOID	USE INSTEAD
INPUT, LINEINPUT	EDIT FIELD
INKEY$	BUTTON, MENU, DIALOG (16)

MACINTOSH APPENDIX

STRUCTURE OF MACINTOSH PROGRAMS

All Macintosh programs are layed-out in essentially the same way. The following simplified outline is of a typical Macintosh type program:

Open Window

Set window size, title and type

Add MENU items and Controls

Create MENU and menu items,
BUTTONS, SCROLL BARS and
EDIT FIELDS as needed.

Initialize Event Trapping

DIALOG ON, ON DIALOG GOSUB
MENU ON, ON MENU GOSUB
MOUSE ON, ON MOUSE GOSUB
TIMER ON, ON TIMER GOSUB
BREAK ON (for debugging)

Main Event Loop

"Main Event Loop"
GOTO "Main Event Loop"
This is an endless loop that just waits for events to occur (if you don't use BREAK ON put a TRONX in here when debugging so you can break-out).

Turn Off Events

So they don't interfere with routines:
DIALOG OFF, MENU OFF, MOUSE OFF,
TIMER OFF or use STOP if you want events to collect in the queue for later.

Routines to Handle Your Events

DIALOG, MOUSE, TIMER and MENU

End

IT'S SUCH A PAIN TO SET-UP MENUS, BUTTONS, EDIT FIELDS AND WINDOWS!!

Well, yes it is. That's why we offer a special ZBasic utility called the "ZBasic CONSTRUCTION SET". This program let's you create all your Controls, Windows and Menus as easily as using MacDraw and then saves your set-up as ZBasic source code that you can merge into your programs. Saves loads of time. Available from Zedcor at 1-800-482-4567 for only $49.95.

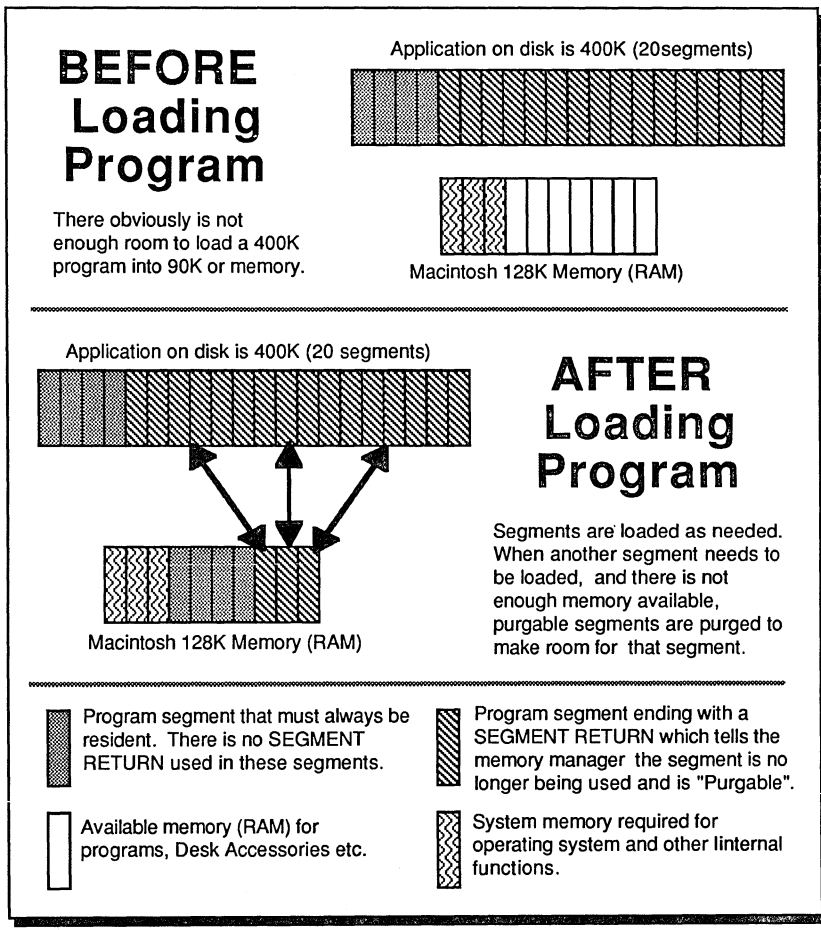
MACINTOSH APPENDIX

MACINTOSH MEMORY MANAGER

USING SEGMENT

Since ZBasic takes advantage of the memory management routines built-in to the Macintosh so can you. If you segment your programs properly, you can create programs that are one megabyte long and execute them on a 512K machine. This is accomplished using the SEGMENT and SEGMENT RETURN statements.

The Macintosh Memory Manager works a little like virtual memory, that is; only the program segments actually needed to be resident in RAM at the same time. Other segments can be left on the disk and will be brought in as necessary:



Note: Segments may vary in size from a few bytes up to a maximum of about 30,000.

MACINTOSH APPENDIX

TAKING ADVANTAGE OF MEMORY MANAGEMENT

If you anticipate a program will be used on systems with limited memory, you can build-in powerful memory management capabilities that will allow large programs to function in limited memory.

- The SEGMENT statement will force a segment break at that point in a program. It is critical that segment breaks do not take place in the middle of loops or structures. System errors may occur in these situations.
- Only programs running on computers with limited memory will show any sign of speed decreases due to more disk accesses. Programs running on systems with plenty of memory will suffer no degradation in performance.
- You may find the size of a specific segment by using the "COMPILE" command. It will print the end and size of each segment, allowing you to predict the sizes of segments being manipulated.
- Use of of the erasable INDEX$ string arrays may also be helpful in memory management.
- Use MEM(n) to find out how much memory is available during runtime.

Important Note: If an attempt is made to load a SEGMENT larger than available memory and there are no more purgable blocks to be erased, a system error will occur.

Important Note: SEGMENT control is only done during RUN* or RUN+. Executing programs from the editor, does not allow the use of SEGMENT or SEGMENT RETURN (a "Segment Overflow Error" will result).

Also see SEGMENT, SEGMENT RETURN and COMPILE.

CHAINING

Chaining on the Macintosh is different than with other versions. The memory management of the Macintosh limits chaining between segments (each segment is limited to about 30K). Therefore when using the standard Chaining syntax be sure to limit each program to one segment.

There are other ways of chaining and passing variables. See the example under WRITE FILE# in this appendix. That example runs other programs and returns with variables intact. Other programs can do the same thing using the READ FILE and WRITE FILE statements.

MACINTOSH APPENDIX

CLIPBOARD

The following examples will allow you to store and retrieve "TEXT" or "PICT" type data from the clipboard:

GET "TEXT" OR "PICT" FROM THE CLIPBOARD

```
CLS
DEFSTR LONG                              :REM REQUIRED FOR CVI("XXXX")
CP&=USR5(0)                              :REM GET HANDLE ZERO LENGTH
T&=CVI("TEXT")
P&=CVI("PICT")
L&=FN GETSCRAP (CP&,T&,D&)               :REM Check for TEXT
LONG IF L& > 0
  GOSUB "PRINT TEXT SCRAP"
XELSE
  L&=FN GETSCRAP (CP&,P&,D&)             :REM Check for PICT
  LONG IF L& > 0
    GOSUB "DRAW PICTURE SCRAP"
  END IF
END IF
:
IF L&=-102 THEN PRINT "EMPTY Clipboard!!"
:
"EXIT"
X=USR6(CP&)                              :REM DISPOSE OF MEMORY BLOCK!
END
:
"PRINT TEXT SCRAP"
LONG IF L&>0
  PRINT STRING$(70,"-")
  ADR&=USR3(CP&)                         :REM GET ADDRESS OF MEMORY
  FOR I& = ADR& TO ADR&+L&-1             :REM BLOCK AND LOCK IT!!!
    A=PEEK(I&)
    LONG IF A <> 13                      :REM TEST FOR END OF LINE (CR)
      PRINT CHR$(A);                     :REM Print TEXT here
    XELSE
      PRINT                              :REM CARIAGE RETURN
    ENDIF
  NEXT : PRINT
  PRINT STRING$(70,"-")
ENDIF
RETURN
:
"DRAW PICTURE SCRAP"
PICTURE (0,0),CP&                        :REM Draw Picture here
RETURN
```

continued...

MACINTOSH APPENDIX

CLIPBOARD continued...

PUT A "PICT" ON THE CLIPBOARD

```
CLS:PICTURE ON
CIRCLE FILL 100,100,50
PICTURE OFF, Picture&
L&=PEEK WORD(PEEK LONG(Picture&)))
Scrap&=FN ZEROSCRAP
Scrap&=FN PUTSCRAP(L&, CVI("PICT"), PEEK LONG(Picture&))
KILL PICTURE Picture&
```

PUT "TEXT" TO/FROM THE CLIPBOARD USING INDEX$

```
CLEAR 1000
FOR I=0 TO 10 : INDEX$(I)=HEX$(I)+STRING$(RND(32),"*") : NEXT I :
N=10
GOSUB "INDEX$ to Scrap" : REM Move INDEX$ to Clipboard
GOSUB "Scrap to INDEX$" : REM Move Clipboard to INDEX$
FOR I=0 TO N : PRINT INDEX$(I) : NEXT
STOP
:
:REM  Move ClipBoard to INDEX$ Array, Returns INDEX$ and N=# of
Strings
:
"Scrap to INDEX$"
H&=USR5(0) : CLEAR 0                        : REM GET HANDLE 0 LENGTH
L&=FN GETSCRAP (H&,CVI("TEXT"),D&)          : REM RESIZES IF REQUIRED
N=0 : IF L& <= 0 THEN "exit"                : REM Return 0 if no scrap
CLEAR L&+512 : A&=USR3(H&) : TEMP$=""       : REM Make Room,Lock Block
FOR I& = A& TO A&+L&-1 :  A=PEEK(I&)        : REM Loop for all Bytes
IF A<>13 THEN T$=T$+CHR$(A) ELSE INDEX$(N)=T$ : N=N+1 : T$=""
NEXT : INDEX$(N)=T$
"exit" :  x=USR6(H&) : RETURN
:
:REM Move INDEX$ to ClipBoard ( N+1 = number of strings to move )
:
"INDEX$ to Scrap" : x=FN ZEROSCRAP : REM Zero Scrap first
L&=MEM(20)      : A&=USR0(L&)    : IF A&=0 THEN BEEP : RETURN
T&=A&           : I&=MEM(40)     : REM T& is temp Ptr,I& points to
INDEX$
FOR I=0 TO N    : sl%=PEEK(I&)   : I&=I&+1    : REM get & Skip Length
byte
IF (T&-A&)+sl%+1 > L& THEN PRINT"Program Error...."      : STOP
BLOCKMOVE I&,T&,sl% :I&=I&+sl% : T&=T&+sl% : POKE T&,13 : T&=T&+1
NEXT I : IF FN PUTSCRAP(T&-A&,CVI("TEXT"),A&) THEN BEEP : BEEP
x=USR1(A&) : RETURN             : REM Dispose of Memory Block
```

MACINTOSH APPENDIX

MACINTOSH GRAPHICS

LOAD AND SAVING MACPAINT FILES

See the example programs for reading and writing MacPaint files under READ FILE# in this appendix. More intensive examples are also included in the Thinner.BAS program included on the ZBasic disk.

TIF FILES

Many users are creating ClipArt with digitizers these days. The latest file type is called "TIF". See the routines in Thinner.BAS for examples of loading this file type.

MACDRAW FILES

MacDraw graphics may be saved to the clipboard or Scrapbook and then loaded into your programs as a PICTURE. This graphic type is called PICT. See examples of loading and saving PICT graphics to resource files under the PICTURE statement in this appendix.

QUICKDRAW ACCESSIBILITY

In addition to the standard graphics commands in the main reference manual and this reference section, this version allows you to access the Macintosh "QuickDraw" routines in ROM.

The end of this appendix contains the toolbox routines and a listing of the "QuickDraw" routines and parameters. There are also a number of examples.

POSTSCRIPT™

See the example program under "Printing" for ways of sending PostScript™ programs to the LaserWriter and other devices.

MACINTOSH APPENDIX

PRINTING

The following pages describe some of the powerful features included with ZBasic that take advantage of the ImageWriter and LaserWriter.

PRINT TEXT AND GRAPHICS TO THE IMAGEWRITER™ AND LASERWRITER™

ZBasic provides some incredibly easy to use graphics and text tools for using the Laserwriter™ and Imagewriter™ printers. Complete integration of text AND graphics is easily accomplished:

ROUTE 128	Route Window text and graphic output to the printer. Graphics can be done with regular ZBasic statements like PLOT, BOX or CIRCLE, or by using PICTURE. Even QuickDraw routines will be routed!
LPRINT @,%	Complete control of where text is printed on the page. "@" controls character position "%" controls pixel position.
COORDINATE	Set up your own coordinate system on the page when using ROUTE 128.
WIDTH LPRINT-2	Disable text wrap around to for extremely high-speed printing.
TEXT	Complete control of FONT, SIZE, MODE and STYLE.
PEN	Control pen width, patterns, styles and modes of lines created with PLOT, BOX and CIRCLE. Even control the patterns used in FILL.
COLOR	Allows you to set the color of the current pen and text for use with ImageWriter printers with color ribbons.
DEF PAGE	Standard Macintosh Page Set-up dialog box.
DEF LPRINT	Standard Macintosh Print dialog box.
PRCANCEL	Checks for DEF PAGE or DEF LPRINT cancel button.
PR HANDLE	Get pointers to number of pages, smoothing, etc., chosen with DEF PAGE and DEF LPRINT.
CLEAR LPRINT	Force printing of text and graphics of current page. If this isn't done the printer driver doesn't know you've finished drawing on the page.
OPEN"C"	Access to direct printing through the serial port. Lets you send control codes to the printer with PRINT# (don't mix with LPRINT).

See the reference section of the appropriate items for complete descriptions of use and example programs.

MACINTOSH APPENDIX

POSTSCRIPT™

You can get even more control of PostScript devices like the LaserWriter™ and LinoTronic™ typesetting equipment by sending them PostScript commands directly. This allows you to get finer line widths and much more control of text positioning and rotating.

The following example program sends a postscript program directly to the LaserWriter:

```
T=0:L=0:B=8192:R=8192:          REM ClipRect Coordinates
WINDOW#1, "PostScript"
:
LONG FN PS$(string$)
  CALL DRAWSTRING(string$)
END FN
:
PICTURE ON
CALL CLIPRECT (T)
CALL MOVETO (20,20)
CALL DRAWSTRING("TextIsPostScript Comment")
CALL PICCOMMENT(190,0,0):       REM  Start PostScript
  CALL PICCOMMENT(194,0,0):     REM  Following text is PostScript
  FN PS$("newpath")
  FN PS$("100 470 moveto")
  FN PS$("500 470 lineto")
  FN PS$("100 330 moveto")
  FN PS$("500 300 lineto")
  FN PS$("230 600 moveto")
  FN PS$("230 200 lineto")
  FN PS$("370 600 moveto")
  FN PS$("370 200 lineto")
  FN PS$("10 setlinewidth"):    REM Try changing 10 to .05
  FN PS$("stroke")
  FN PS$("/Times-Roman findfont 12 scalefont setfont")
  FN PS$("230 600 moveto")
  FN PS$("(Hello World) show")  REM This text prints upside/down
CALL PICCOMMENT (191,0,0) :     REM End Postscript output
PICTURE OFF, MyPic&
DEF LPRINT
ROUTE 128
PICTURE, MyPic&
CLEAR LPRINT
ROUTE 0
END
```

PostScript is a Page Description language from Adobe Systems Incorporated. They have several books out about PostScript that are very useful and easy to read (we use them). They are available from most bookstores.

POSTSCRIPT Language Reference Manual
Published by Addison Wesley

POSTSCRIPT Language Tutorial and Cookbook
Published by Addison Wesley

MACINTOSH APPENDIX

APPLETALK™

ZBasic offers the user access to the Appletalk network. The network lets you "Talk" to other computers, printers and other connected storage devices.

The terms used in these statement and function definitions are explained in the Appletalk section of "Inside Macintosh". "Inside Macintosh" is essential to understanding the Appletalk Network system.

OPEN NETWORK

Initializes Appletalk and configures printer port.

OPEN SOCKET *Socket, "OBJECT", "TYPE", "ZONE" [, [TryCount] [, [TryInterval] [, [Network] [, [Node] [,Socket]]]]]*

Opens a socket for purpose of getting requests from other computers or "Nodes" on the network.

NETWORK DIRECTORY *"OBJECT", "TYPE", "ZONE", Maxnames%, Network%(), Node%(), Sockets%(), Object$(), Type$(), Zone$() [, [count] [,interval]]*

Shows other sockets in other nodes on the network.

GETREQUEST *Socket, Synch%, CompletionRoutine, RequestBufferLen%, RequestBuffer^, ATPflags%, UserData&, Network%, Node%, Socket%, BitMap%, Transaction%*

Reads a request from another node.

SENDREQUEST *Synch%, CompletionRoutine, RequestBufferLen, RequestBuffer^, Response%, Size%(), Buffer&(), Bytesgot%(), Userdata, Network, Node, Socket, Flags%, Bitmap%, Transid%, [, [count%] [,interval%]]*

Sends a request to another node and waits for a response.

SENDRESPONSE *Socket, Synch%, CompletionRoutine, ATPflags, Network, Node, Socket, Buffers, Buffersize%(), Bufferptr&(), Userdata&(), Transid, Userdata&*

Replies to a request from another node.

MACINTOSH APPENDIX

MACINTALK™

ZBasic supports the MacinTalk voice synthesizer software from Apple. It is included on the disk and should be copied to the system folder of any disk that uses this application.

Be sure to see the following statements in the reference section of this appendix for details:

OPEN TALK statement
TALK statement

The following information is for serious "MacinTALKERS".

ENDING A SENTENCE

End a sentence with a period (.), exclamation mark (!), or a question mark (?).

The period causes a final fall in pitch at the end of a sentence. The question mark will cause a rise in pitch for yes or no questions.

Note that the question "How much do you weigh" would not require a question mark. Always CLOSE Macintalk to avoid system errors.

SPELL IT LIKE IT SOUNDS WITH THE ENGLISH TEXT READER

If using the default English text reader, remember that certain letters may be better than others.

For instance the word "bats" might sound better with Macintalk if you use "batz" instead. "Michael" comes out like "Mitch-Ale". Use "Mikil" instead. Listen carefully to the sound of words and try typing them in as they sound NOT as they're spelled.

USING PHONEMES TO PRODUCE SPEECH

To create more natural sounding words and sentences you may want to use Phonemes instead. This is accomplished by using "no Reader" when opening Macintalk with OPEN TALK, #n, "noReader".

The next page contains a chart for more information about phonemes, dipthongs, stress and contractions.

HOW TO GET A MACINTALK LICENSE FROM APPLE

If you are selling your programs, you may not distribute the MacinTalk™ file without the specific written permission of Apple Computer, Inc.

Licenses which permit distributing the latest MacinTalk™ software are available for a moderate annual fee.

For details, contact Apple's Software licensing department at: (408) 973-4667

MACINTOSH APPENDIX

MACINTALK PHONEME TABLE ("noReader")

VOWELS

IY	beet	IH	bit	EH	bet	AE	bat
AA	hot	AH	under	AO	talk	UH	look
ER	bird	OH	border	AX	about	IX	solid

NOTE: AX and IX should never be used in stressed syllables.

DIPTHONGS

EY	made	AY	hide	OY	boil	AW	power
OW	low	UW	crew				

CONSONANTS

R	red	L	yellow	W	away	Y	yellow
M	men	N	men	NX	sing	S	sail
SH	rush	F	fed	TH	thin	Z	has
ZH	pleasure	V	very	DH	then	CH	check
J	judge	/H	hole	/c	loch	B	but
D	dog	T	toy	G	guest	K	camp

SPECIAL SYMBOLS

DX	pity (Tongue flap)	Q	kitt_en (glottal stop)
RX	car (postvocalic R and L)	LX	call
QX	silent vowel		

CONTRACTIONS

UL=AXL	IL=AXL	UM=AXM	IM-IXM
UN=AXN	IN=IXN		

STRESS MARKS (digits 1-9)

Some typical stress values:

5	nouns	2	pronouns	4	verbs
5	adjectives	7	adverbs	7	quantifiers
9	exclamations	Ø	articles	Ø	prepositions
Ø	conjunctions	1,2	Secondary stress		

PUNCTUATION

.	Sentence terminator	?	Sentence terminator
-	Phrase delimiter	,	clause delimiter
( )	noun phrase delimiters		

MACINTOSH APPENDIX

CONVERTING MSBASIC™ PROGRAMS

CONFIGURING ZBASIC FOR EASIER CONVERSION

Here's how to configure ZBasic to make MSBASIC™ programs easier.

Under the "Configure" menu select "Change Configuration" and set the following parameters:

Spaces Required after Keywords	YES
Convert to Uppercase	YES
Optimize Expressions as Integer	NO*
LOCATE Y,X:	YES (ZBasic does x,y)
Test Array Bounds	YES**
Test String Lengths	YES**

* May slow down execution more than necessary. See "Math" in front of this manual.
** Enable these checks only during debug stage for best program speed and size.

MAIN DIFFERENCES BETWEEN ZBASIC™ AND MSBASIC™

The biggest differences between ZBasic and MSBasic™ occur with graphics and disk commands.

The following pages will cover most of the MSBASIC commands not supported by ZBasic and suggestions for conversion. It is important to note these differences because there are some commands that are the same but we have added certain extensions to them to give you more power.

STRINGS

The maximum string length for ZBasic strings is 255 characters. MSBASIC allows up to 32,767 characters so any programs using strings that exceed the 255 limit must be modified by the user. The usual reason for needing more than 255 is with EDIT FIELD and text editing applications. See ZBasic's TEHANDLE function and the Toolbox TEXT EDIT section for ways of overcoming this.

GRAPHICS

ZBasic incorporates a Device Independent Graphics system. This means that graphics created in windows correspond to the standard coordinate system of 1024x768. Note that this version of ZBasic allows you to reset these coordinates. The command is COORDINATE Xmax, Ymax.

It is normally to your advantage to use the device independent graphics as this is what is used in the main ZBasic reference manual. To convert existing programs it is useful to revert to the MSBASIC Pixel coordinate system. This may be done for each window easily by using the ZBasic COORDINATE WINDOW statement.

MACINTOSH APPENDIX

MSBASIC™ LPRINT *VERSUS* ZBASIC™ LPRINT

When you use the LPRINT statement in ZBasic, text is routed through the Macintosh printer driver. This lets change text styles and such with the TEXT statement. Some versions of MSBASIC send output directly to the serial port. These programs may even using control codes and such which are stripped out when sent to the Macintosh printer driver.

To send output directly to the printer driver use OPEN"C",-2, 9600. See example under OPEN"C" in this appendix (also see HANDSHAKE).

Note that in most cases this is not needed. Just put a WIDTH LPRINT-2 at the beginning of your program and set the text characteristics with TEXT and away you go. Not only is the quality much better but the speed is nearly as good.

You could even select DRAFT printing from the DEF LPRINT window to get faster printing in some cases.

ZBASIC EVENT TRAPPING

ZBasic checks for events at the beginning of lines that are physically between lines containing EVENT ON and EVENT OFF (events being: DIALOG, MENU, MOUSE, TIMER and BREAK).

MSBASIC checks the events according to the program flow between the EVENT ON and EVENT OFF.

THINGS TO PUT AT THE FRONT OF A MSBASIC™ PROGRAMS YOU'RE CONVERTING

COORDINATE WINDOW	Set to "Pixel" coordinates like MSBASIC uses.
DEFMOUSE=-1	Set to MSBASIC mouse commands. To use ZBasic mouse statements see "MOUSE" in the main reference section.
WIDTH LPRINT-2	Disregards characterwrap to the printer. Increases printing speeds by 2-5 times.
WIDTH -2	Disregards character wrap to the window and increases printing speed to the screen a bit.
TEXT 1,12,,0	Sets text font, size and mode to MSBASIC defaults.

MACINTOSH APPENDIX

MSBASIC™	ZBASIC™
CALL	CALL with parameter passing is supported but libraries are not. See "For Machine language programmers" in the beginning of this appendix. Note that MSBASIC and ZBasic floating point are stored differently in memory.
CDBL(x)	ZBasic converts for you automatically.
CHAIN	See "Memory Management" in this appendix.
CINT(x)	See CINT function in the MSDOS appendix.
CHDIR	See FILES$ and DIR in this appendix.
CIRCLE (x,y),radius	CIRCLE x,y,radius Note: ZBasic uses BRADS instead of Radians for start and end of arcs.
CSNG(x)	ZBasic converts for you automatically.
CVDBCD, CVSBCD	See CVB
CVS, CVD	See CVB
ERASE	SEE MEM(-1), CLEAR nnnn [index#] for erasing INDEX$ arrays.
ERL	Not supported
ERR	Not supported
FIELD	See READ# and WRITE#
FILES	Use DIR and FILES$ in this appendix.
GET #1,	See READ# and RECORD#.
IGNORE$	Meta commands not supported.
INCLUDE$	Meta commands not supported.
INPUT$	Not supported. See READ#
LBOUND, UBOUND	Not supported.
LINE (x1,y1)	PLOT TO x1,y1
LINE (x1,y1)-(x2,y2)	PLOT x1,y1 TO x2,y2
LINE (x1,y1)-(x2,y2),,B	BOX x1,y1 TO x2,y2
LOF	Different. Use LOF(filenumber,1) to do the same thing. See LOF
LPOS	See POS(1)
LSET	Not supported. Use: A$=STRING$(" ", length-LEN(B$))+B$
MERGE	Command only. May not be used in a program. Use RUN+
MKS$, MKD$	See MKB$
MKSBCD$, MKDBCD$	See MKB$
ON ERROR	DISK ERRORS ONLY with ZBasic. See DISK ERRORS under "Files".
OPTION BASE	Pull down the "CONFIGURE" menu and set to this option.
OPTIONS$	Meta commands not supported.
PAGE$	Meta commands not supported.
PEEK	Use LongIntegers. See also PEEKWORD and PEEKLONG
PICTURE$	DIFFERENT. ZBasic uses PICTURE , longint&
POKE	Use LongIntegers.. See also POKEWORD and POKELONG
PRESET (x,y),color	COLOR=color:PLOT x,y. Step is not supported.
PRINT USING	Same except for string formatting.
PSET (x,y),color	COLOR=color:PLOT x,y
PUT#1,	See WRITE# and RECORD#.
RESTORE	ZBasic positions to the DATA ITEM instead of LINE number.
RESUME	Not supported.
RND(x)	Different. ZBasic returns a number between 1 and x, not Ø and 1.
RSET	Not supported. Use: A$=B$+STRING$(" ",length-LEN(B$))
SHARED	See DIM and CLEAR END
SUB	See LONG FN, APPEND line and SAVE+ in the reference section.
WIDTH	Slightly different. See WIDTH.
WIDTH#	Not supported. See ROUTE filenumber
WINDOW OUTPUT#	Not supported. Use ROUTE 128 to route all window text and most graphic activity to an imagewriter or laserwriter.
WRITE#	Different. Use ZBasic's PRINT# or WRITE# to convert

MACINTOSH APPENDIX

USING RMaker™

About this Chapter

This chapter describes RMaker, an application licensed from Apple Computer, Inc. we've included with ZBasic™ as a convenience. It is used to produce and integrate resource files into your programs.

About RMaker™

RMaker is the Macintosh 68000 Development System's (MDS) Resource Compiler. Primary use with ZBasic is to allow you to include resources and icons with you applications.

RMaker takes text file(s) as input and produces a resource file. The text file contains an entry for each resource, as described below. These entries can specify all information necessary to define the resources , or they can cause existing resources to be read from other files.

For example, during program development you'll typically use separate application and resource files. Once the application is finished, you should combine the files. Simply use the INCLUDE statement to read in the application created by ZBasic™. It is already stored as resources of type 'CODE'.

RMaker™ Input Files

An RMaker input file is a text file that may be created using MDS EDIT (or ZBasic's editor if saved in ASCII without line numbers). By convention, RMaker text files have the extension .R.

RMaker ignores all comment lines and blank lines (except in some cases a blank line may be required). It also ignores leading and embedded spaces (except lines described as strings). Comment lines begin with an asterisk . To put comments at the end of other RMaker lines precede the comment with two consecutive semicolons (;;).

Naming the Resource File

The first nonblank and noncomment line of the input file specifies the name of the resource file to be created. If the filename has the extension .REL, a file is generated that can be linked using the MDS linker (normally not applicable with ZBasic). If the file is to be an application, it should have no extension. If not, the file will be a resource file and should have the extension .Rsrc. The line following the resource's filename should either specify the file type and creator bytes for the FINDER or be blank. For example, the two lines

```
NewResFile.Rsrc
PNTGMPNT
```

specify the file named NewResFile.Rsrc as the output file, and the bytes PNTGMPNT as the type and creator bytes. These bytes tell the FINDER that the file is a 'Paint' file created by MacPaint (the finder will try to launch MacPaint if you select this file).

More typically, these two lines will look like this:

```
MyApplication
APPLMYAP
```

This designates the file MyApplication as the output file. The file is an application (type 'APPL') and creator type of 'MYAP'. If you do not set a value for these bytes they are set to '0' (zero).

Appending to an Existing Application

If you wish to add the resources defined in your input file to those in a ZBasic file, simply precede the filename with an exclamation mark (!). For example

```
!OldResFile.Rsrc
```

tells RMaker to add the resources to the file OldResFile.Rsrc

MACINTOSH APPENDIX

Adding Resources

The rest of the resource file consists of INCLUDE statements and "type statements".

INCLUDE statements are used to read in entire resource files. An INCLUDE statement looks like this:

```
INCLUDE filename
```

Type statements consist of the word "TYPE" followed by the resource type and, below that, one or more resource definitions. The resource type must be capitalized to match a predefined resource type (as defined on the next page). This statement creates three resources of type 'STR'.

```
TYPE STR
       ,1
This is a string
       ,2
Gnirts a si siht
       ,3
Hits is a grints
```

It is not necessary for all resources of a given type to be declared together; however, all resources of a type must have unique resource IDs. If you specify a resource ID that is already in use, the new resource replaces the old one. A resource looks like this:

[resource name], resource ID [(resource attribute byte)]
type-specific data

The square brackets indicate the resource name and resource attribute byte are optional. The comma before the resource ID is mandatory. The default attribute is '0'. Here are some resource definitions:

```
TYPE STR
NewStr ,4 (32)
This resource has a name and an attribute byte!!
       ,5
This one has only a resource ID
MyNewStr, 6
This one has a name and a resource ID
```

The type specific data is different for each resource type. The type specific data for a 'STR' resource is simply a string. The next page describes the type-specific data for resource types defined by RMaker.

Syntax of RMaker lines

There are just a few general rules to lines read by RMaker.

- Leading and embedded blanks are ignored, except when necessary to separate multiple numbers on a line, or when they are a part of a string.
- Numbers are decimal, unless specified otherwise.
- RMaker is sensitive to line breaks. Thus, if a type description shows 4 values on a line, you must put 4 values on that line.

Special Symbols

Two special symbols can be used in resource definitions: The continuations symbol (++) and the enter ASCII symbol (\)

++	goes at the end of a line that is continued on the next line.
\	precedes two hexadecimal digits. The ASCII character is entered into the resource definition.

Look at the description of the STR resource type for examples of this special symbol. As previously mentioned, blank lines are ignored. To enter a blank line that isn't ignored use, use \20.

MACINTOSH APPENDIX

DEFINED RESOURCE TYPES

RMaker has 12 defined resource types:

ALRT	**BNDL**	**CNTL**
DITL	**DLOG**	**FREF**
GNRL	**MENU**	**PROC**
STR	**STR#**	**WIND**

The format of the type-specific data for each type is shown by example, below. The type 'GNRL' is used to define your resource types. It is explained later.

ALRT Templates

```
TYPE ALRT
   ,128                    ;; Resource ID
50 50 250 250              ;; top left, bottom right
1                          ;; resource ID of item list
7FFF                       ;; stages word in hexadecimal
```

BNDL Application Bundle

```
TYPE BNDL
   ,128                    ;; Resource ID
MPNT 0                     ;; bundle owner (Macpaint)
ICN#                       ;; resource type
0 128 1 129                ;; local ID 0 maps to resource 128; 1 to 129
FREF                       ;; resource type
0 128 1 129                ;; local ID 0 maps to resource 128; 1 to 129
```

CNTL Control Template

```
TYPE CNTL
   ,130                    ;; resource ID
Stop                       ;; title
244 40 260 80              ;; top left, bottom right
Invisible                  ;; see note
0                          ;; ProcID (control definition ID)
0                          ;; RefCon (reference value)
0 1 0                      ;; minimum maximum value
```

Note: Controls can be defined to be Visible or Invisible. Only first character (V or I) is significant.

DLOG Dialog Template

```
TYPE DLOG
   ,3                      ;; resource ID
This is a Dialog box.      ;; message
100 100 190 250            ;; top left, bottom right
Visible GoAway             ;; box status ( see note)
0                          ;; procID (dialog identification ID)
0                          ;; refCon (reference value)
129                        ;; ID of item list ('DITL', above)
```

Note: A Dialog box can be **Visible** or **Invisible**. **GoAway** and **NoGoAway** determine whether or not the Dialog box has a close box. Only the first characters (**V, I G, N**) are significant.

MACINTOSH APPENDIX

DITL Dialog or Alert Item list

```
TYPE DITL                       ;; resource ID
   ,129                         ;; 5 items in list

StaticText                      ;; static text dialog item (see note)
20 20 32 100                    ;; top left, bottom right
Whooppie                        ;; message

EditText                        ;; Editable text dialog item (see note)
20 120 32 200                   ;; Top left, bottom right
Default message                 ;; message

RadioButton                     ;; radio button dialog item (see note)
40 40 60 150                    ;; top left, bottom right
Hello                           ;; message

Checkbox Disabled               ;; disabled dialog item (see note)
75 40 95 150                    ;; top left, bottom right
Goodbye                         ;; message

Button                          ;; button dialog item (see note)
75 160 95 200                   ;; top left, bottom right
Hi!                             ;; message
```

Note: RMaker recognizes the following words as DITL items:

BtnItem	**Button**	**Check**	**Checkbox**	**ChkItem**
Ctrl	**CtrlItem**	**Edit**	**EditText**	**Icon**
IconItem	**Pic**	**PicItem**	**Radio**	**RadioButton**
RadioItem	**ResCItem**	**ResCtrl**	**Stat**	**StaticText**
StatText	**User**	**UserItem**		

These items are assumed to be enabled. Otherwise you must specify the entire word **'Disabled'**.

FREF File Reference

```
TYPE FREF
   ,128                         ;; resource ID
APPL 0                          ;; file type, local ID of icon

   ,129                         ;; resource ID
TEST 127 myFile                 ;; file type, local ID of icon, filename
```

Note: If there is no filename it can be omitted.

MENU

```
TYPE MENU
   ,3                           ;; resource ID
Transfer                        ;; menu title
Edit                            ;; item 1
Asm                             ;; item 2
Link                            ;; item 3
(-                              ;; item 4 (draw a line)
Exec                            ;; item 5
                                ;; MUST be followed by a blank line!!
```

MACINTOSH APPENDIX

PROC Procedure

```
TYPE PROC
   ,128                        ;; resource ID
MyProcedure                    ;; filename
```

This type is to create resource that contain code. It reads the first code segment from an application file (the 'CODE' resource with ID=1), strips the first four bytes off of it (used by the segment loader), and saves it as a resource of type 'PROC'. It is useful for defining code types such as 'DRVR', 'WDEF', and 'PACK'. An example is given below in the section for creating your own resource types.

STR Strings

```
TYPE STR                       ;; 'STR ' (space required at end STR_)
   ,1                          ;; resource ID
this is a string               ;; and a string

   ,23                         ;; resource ID
This is a string++             ;; and a long string
that shows the line++
continuation characters.

   ,25 (32)                    ;; resource ID, optional attribute byte
I've got attributes!           ;; and a string

   ,27                         ;; resource ID
Testing, \31, \32, \33         ;; 'Testing, 1,2,3' the hard way
```

STR# A Number Of Strings

```
TYPE STR#
   ,1                          ;; resource ID
4                              ;; number of strings
This is string one             ;; and the strings...
string two
and string three
and the last number 4
```

WIND Window Template

```
TYPE WIND
,128
Wonder Window                  ;; Window title
40 80 120 300                  ;; top left, bottom right
Invisible GoAway               ;; Window Status (see note)
0                              ;; ProcID (Window definition ID)
0                              ;; RefCon (reference value)
```

Note: A window can be **Visible** or **Invisible**; **GoAway** or **NoGoAway** determine whether or not the window has a close box. Only the first character of each option is significant **(V, I, G, N)**.

Creating Your Own Types (GNRL)

There are two ways to create your own resource types. The first is to equate a new type to an existing type. For example, you can create a resource of type 'DRVR' like this:

```
TYPE DRVR=PROC                 ;; Type 'DRVR' is just like 'PROC'
   ,17 (32)                    ;; resource ID, Attribute byte
MyDriver                       ;; filename
```

The file MyDriver should be a single-segment application created by ZBasic. Recall that the 'PROC' type reads in the resource of type 'CODE' with ID=1; then strips off the header bytes.

The other way to create your own type is to equate the new type to 'GNRL' and then to specify the precise format of the resource. A set of element type designators lets you define the type of each element that is to be placed in the resource.

MACINTOSH APPENDIX

Here are the element type designators:

.P Pascal string (same as ZBasic strings)
.S String without length byte
.I Decimal Integer
.L Decimal Longinteger
.H Hexadecimal

.R Read resource from file. .R is followed by: filename type ID

For example, to define a resource of type 'CHRG' consisting of the integer 57 followed by by the Pascal string 'Finance charges', you could use the following type assignment:

```
TYPE CHRG=GNRL              ;; define type 'CHRG'
,200                        ;; resource ID
.I                          ;; decimal integer
57
.P                          ;; a Pascal string (same as ZBasic strings)
Finance charges
```

A more practical example: An application that has its own Icon must define an icon list and reference it as 'FREF' (described above). Such an icon list can be described as follows:

```
TYPE ICN#=GNRL              ;; icon list for application
   ,128                     ;; resource ID
.H                          ;; enter two icons in hexadecimal
0001 0002 0003 0004         ;; each is 32 bits by 32 bits
   ....
007D 007E 007F 0080         ;; for 128 words total
```

The .R type designator is used to include an existing resource as part of a new resource type. For example, to read an existing 'FONT' resource into a new resource of type 'FONT', use the following resource definition:

```
TYPE FONT=GNRL              ;; define a new type
   ,268                     ;; resource ID
.R                          ;; read from the System File
System FONT 268             ;; the 'FONT' resource with ID 268
```

Using RMaker™

Once you have created the input file to RMaker, the hard work is done. Simply select and open the application '**RMaker**'. The standard file selection window is automatically opened. Select the file you want to compile and off it goes.

By default, the standard file selection window displays all the text files on the disk. If you want to display only .R files, cancel the selection window and, choose .R Filter from file menu, then chose Compile from the the file menu to re-display the file selection window.

When RMaker is compiling a file, the name of the source file is displayed in the upper left of the window, and the name of the output file is displayed in the upper right.

As the file is compiled, the current size of the resource data, the size of the resource map, and the total size are tracked on the right half of the screen. In addition, as each line is compiled, it is displayed on the screen.

If there are no errors in the RMaker file, a resource file with the specified name is created.

MACINTOSH APPENDIX

RMaker™ ERROR MESSAGES

If an error occurs, the line containing the error is the last line on the screen. RMaker then displays a box with an error message in it. A brief description of the error message is included for those that are not self explanatory:

An Input/Output error has occurred	
Bad attributes parameter	
Bad bundle definition	
Bad format number	
Bad format resource designator in GNRL type	*User-defined resource type error.*
Bad ID number	
Bad item type	
Bad object definition	*Happens if specified file is of wrong type.*
Bad type or item declaration	
Can't add to the file--- disk protected or full?	
Can't create the output file	
Can't load INCLUDE file	
Can't open the output file	
Out of memory	
Syntax error in source file	
Unknown type	*The specified resource type is not defined.*

Error note: When an alert occurs, the last line displayed usually contains the error. If a type declaration is wrong, a second line is displayed.

OTHER NOTES ON RMaker™

Bundle bit RMaker will automatically set the Bundle Bit on the output file if it has processed a BNDL resource from the source file.

Restrictions in Resource File Names
RMaker will not accept resource names that contain a comma (,) or the substring 'TYPE'. These are treated as reserve words in RMaker resource files.

New Flags: /Quit and /NoScroll
Two new flags, /Quit and /NoScroll, have been added to RMaker.

- The /Quit flag is provided when the normal Exec process is not used. If a /Quit flag is encountered, RMaker will quit to the Finder after finishing the file rather than waiting for another file to be selected.

  Use the /Quit flag when the source file is passed via the application parameters without the path manager (for example, opening RMaker with an RMaker source file from the Finder). If RMaker is called as part of the Exec, the /Quit flag is unnecessary, since the Exec will continue to execute after RMaker is finished.

- The /NoScroll flag suppresses scrolling in the left pane of the window. To speed the compile, only Type lines are displayed. If an error is found, the line that contains the error is displayed.

  The /NoScroll and /Quit flags must be the first lines in the file, before the destination filename. Therefore, you cannot name an output file /Quit or /NoScroll.

MenuKey Equivalents
MenuKey equivalents are supported and can be changed with a resource editor, if desired. NOT FOR USE WITH ZBASIC.

MACINTOSH APPENDIX

HOW TO PUT ICONS IN YOUR ZBasic™ APPLICATIONS

A lot of you have requested a simple, by the number, example of installing icons in your applications.

This example sequence is easy and everything you need, except your program, is included. We've even included an icon resource file with icons for your application and your applications data files.

1. Write your program. Don't worry about creating your own icons until your program is running flawlessly.

2. From ZBasic load your program. Sect the "Configure" menu and select "Change Configuration".
   Set the "Application Bundle bit" to "YES". This tells the Finder that your program will have its own Icon(s). Set file Creator to "MIKE". This tells the finder that MIKE is the CREATOR. Set Default Data File Type to "ANDY". This will be the application FILE TYPE. Of cource you may change these names but we will refer to them again as MIKE and ANDY.

   Note: If your application is to be commercialized you should get approval for (and reserve) your Creator and Data type from Apple Computer, Inc. at (408) 973-4667.

3. Now Compile your program by typing in RUN* or RUN+. Give your program the desired filename.

4. "QUIT" ZBasic and load ResEdit, IEDIT or some other resource/icon editor. There are quite a few of them available in the public domain. Load "XICON.RSRC" ino your icon editor and modify our default icons according to your preference. The example is a simple question mark icon that you can modify for your needs . You may also add new icons to this file for use in your menus or windows. Notice in our example program, "IOTEST.R", that Icon numbers 128 and 129 are the only ones specified. If you add new icons with different numbers, make the changes in the "IOTEST.R" file.

   Edit the icon as you like using your icon editor and save it out using the same or another filename with .RSRC at the end of it. The RSRC is the convention for resource files.

MACINTOSH APPENDIX

ANY TEXT EDITOR

5. Load the example RMaker source file: IOTEST.R into the text editor of your choice.

   You will want to change boldfaced items:

```
* EXAMPLE RMAKER FILE
* Illustrates how to add resources to your applications.
*
!IOTEST                    ;; Your filename with "!" in front

INCLUDE XICON.RSRC         ;; Use your resource/icon filename

TYPE FREF                  ;; FILE REF INFO
 ,128                      ;; APPL FILE ICON# IN XICON.RSRC
APPL 0                     ;; APPL IS TYPE 0
 ,129                      ;; DATA FILE ICON# IN XICON.RSRC
ANDY 1                     ;; Use your Data type if applicable

TYPE BNDL
 ,128
MIKE 0                     ;; Use your CREATOR type
ICN#                       ;; TYPE0=ICON#128, TYPE1=ICON#129
0 128 1 129                ;; ICON # DEFINITIONS
FREF                       ;; TYPE0=ICON#128, TYPE1=ICON#129
0 128 1 129                ;; FILE REF DEFINITIONS

* NOTE: Change icon numbers if yours are different.

TYPE MIKE= GNRL
 ,0
.P
```

MODIFY AS NEEDED

CHANGE:	TO:
"!IOTEST"	Your application filename with an "!" in front of it.
"XICON.RSRC"	The filename of your icon reources
"ANDY"	Your DATA file type
"MIKE"	Your CREATOR file type
128	Your icon number if applicable
129	Your icon number if applicable
Icon numbers	If you are using more than two icons.

Save the edited "IOTEST.R" using SAVE AS: "ICONTEST.R" , or whatever filename you prefer.

RMaker

6. Load RMaker. Load modified "ICONTEST.R" above or whatever filename you chose. RMaker will compile the resources and add them to your application. Your program will now have its own icon(s). That's all there is to it!

MACINTOSH APPENDIX

Notes

MACINTOSH APPENDIX

Macintosh™ Reference Section

An alphabetical listing of the
statements, functions, desk accessories and commands
that are not in the Standard Reference section
or have added or different syntax.

MACINTOSH APPENDIX

APPEND statement

FORMAT **APPEND** *[#] filenumber*

DEFINITION Moves the file pointer to the end of the file specified by filenumber so that subsequent PRINT#, WRITE# or ROUTE# data may be appended to that file.

The file specified by *filenumber* must already be open.

EXAMPLE

```
DEFOPEN "TEXT"
FOR I=1 TO 4
  PRINT I
  OPEN"R",1,"Testfile"
  APPEND #1
  PRINT #1,"HELLO",I
  CLOSE 1
NEXT
```

REMARK Always use this statement BEFORE you do any writes.

See OPEN"A", CLOSE, READ, WRITE, PRINT#, INPUT# and the section called "Files".

Note: This statement is not related to the standard ZBasic APPEND command. See APPEND command in the main reference section for details.

MACINTOSH APPENDIX

APPLE MENU statement

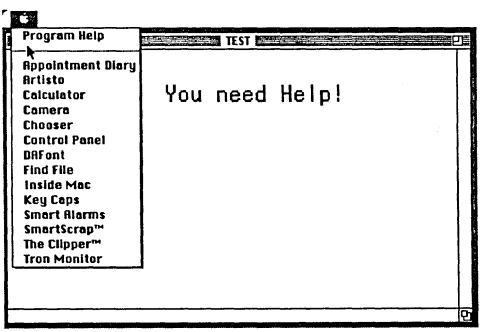
FORMAT **APPLE MENU** *simplestring*

DEFINITION Allows you to put a menu item under the menu. *simplestring* sets the name of the menu item. If MENU ON is active and the item is selected, the program will branch to the routine specified by the line number or label used in ON MENU GOSUB.

To determine if the menu has been chosen, the MENU(Ø) function will return 255 and MENU(1) will return 1. Both functions must be used and in that order.

EXAMPLE

```
WINDOW#1, "TEST": TEXT ,24:CLS
APPLE MENU "Program Help"
:
FLUSHEVENTS:BREAK ON: ON BREAK GOSUB "END"
MENU ON: ON MENU GOSUB"Menu"
:
"Main Event Loop"
GOTO "Main Event Loop"
:
MENU OFF:BREAK OFF
"Menu"
X=MENU(0):Y=MENU(1)
IF X=255 THEN PRINT TAB(12);"You need Help!"
MENU: RETURN
"END": TEXT ,9: END
```



REMARK Meta-characters may be used to format the appearance of the item or to include icons. See the MENU statement and function for more information about meta-characters and using the Apple Menu.

Also see SCROLL BUTTON for an example of scrolling through text files which you might want to use for help or instructions from this menu item. See PICTURE for ways of saving PICTURE handles that may be used from your Apple Menu.

Note: APPLE MENU may only be executed once in a program. You cannot change the menu contents once it is executed except with the toolbox.

MACINTOSH APPENDIX

BLOCK MOVE statement

FORMAT **BLOCK MOVE** *source&, destination&, byte*

DEFINITION Allows you to move large blocks of memory from one place to another. Note that *Source&* and *Destination&* parameters require LongInteger numbers or variables (*byte* will require a LongInteger variable if the value is greater than 32,767).

EXAMPLE1

```
REM Program will load and display a MACPAINT file
DIM X%(1),71 X$(719) : X%(0)=576 : X%(1)=720
:
A$=FILES$(1,"PNTG",,V%): IF A$="" THEN STOP
OPEN"I",1,A$,1,V% : FL&=LOF(1)-512 : RECORD #1,512
:
A&=VARPTR(A$) : Y&=VARPTR(X$(0)) : X&=A& : N=256
FOR I=1 TO 720 : REM Lines in a MACPAINT Picture
  LONG IF N>180
    BLOCKMOVE X&,A&,256-N : X&=A&
    IF N>FL& THEN NX=FL& ELSE NX=N
    READ FILE #1,A&+256-N,NX : FL&=FL&-NX
  END IF
  REM Puts bit image in memory.
  CALL UNPACKBITS(X&,Y&,72):N=X&-A&
NEXT
CLOSE#1
PUT (0,0),X%(0),PSET: REM PUT PAINT picture on screen.
END
```

EXAMPLE2

```
REM Blockmove the contents of one variable into another
A$="HELLO":B$=""
BLOCKMOVE VARPTR(A$), VARPTR(B$), LEN(A$)+1
PRINT B$

RUN

HELLO
```

REMARK

Warning: Indescriminate use of this statement is guaranteed to produce system bombs.

MACINTOSH APPENDIX

BREAK statement

FORMAT

BREAK ON
BREAK OFF
BREAK STOP

DEFINITION

Allows the user to enable or disable checking for <COMMAND PERIOD>. This allows a user to <BREAK> out of a program and return to the ZBasic edit modes (or to the system if encountered in a stand-alone application).

EXAMPLE

```
WINDOW#1, "TEST"
:
PRINT"Press <COMMAND PERIOD> to exit this program"
:
FLUSHEVENTS: BREAK ON:ON BREAK GOSUB "Break out"
:
"Main Event Loop"
GOTO "Main Event Loop"
:
BREAK OFF
:
"Break out"
END
```

REMARK

Also see TRON and TROFF in the main reference section and TRON#128, TRONV and TRON MONITOR in this appendix for other important debugging tools.

Note: With ZBasic; BREAK STOP and BREAK OFF have the same affect.

Important Note: Events are trapped physically between the BREAK ON and the BREAK OFF at the beginning of each line. Any lines of codes lying outside of this area do not check for BREAK events. Note that the following line would result in an endless loop and BREAK would not function since the line never gets back to the beginning of the line:

```
 BREAK ON:
*FOR X= 0 TO 10 STEP 0:NEXT
 END
```

Note: ON BREAK GOSUB is optional. If it is not used with BREAK ON an END will be done when the <COMMAND PERIOD> keys are pressed between the BREAK ON and BREAK OFF (except as noted in example).

MACINTOSH APPENDIX

BUNDLE command

FORMAT **BUNDLE** status

DEFINITION Used to set the bundle bit for stand-alone applications you create. This must be set to allow icons to be added or for use with FINDERINFO.

This is useful for determining what to do with files that your application must execute, print or load when the status is passed from the finder with FINDERINFO to a stand-alone application.

STATUS	Function
Ø	Clears the BUNDLE bit
NOT Ø	Sets the bundle bit to ON

EXAMPLE

```
REM*************************************************************
REM*  Example FINDERINFO program created  5/10/87    A.G.    *
REM*  For This example to work the bundle bit must be ON     *
REM*  and the file TYPE and CREATOR must be Valid & Unique   *
REM*  Program creates programs which can be double-clicked   *
REM*  and will cause this program to execute & load them.    *
REM*************************************************************
DEFSTR LONG : DIM 31 FileName$(9),Volume%(9),Type&(9)
Count%=10    : REM MAX # OF FILES ACCEPTED
Boolean%=FINDERINFO(Count%,FileName$(0),Type&(0),Volume%(0))
IF Boolean% THEN PRINT "Print File..." ELSE PRINT "Load File..."
LONG IF Count%
  PRINT "File Name","File Type","Volume #"
  FOR I=1 TO Count%
    Type$=MKI$(Type&(I-1))
    PRINT "'";FileName$(I-1);"'",Type$,Volume%(I-1)
  NEXT
XELSE
  INPUT "No Files received, Create some? Y/N:";C$
  C$=UCASE$(C$) : IF C$="N" OR C$="" THEN STOP
  FOR I=1 TO 3 : REM Create 3 Different Files
    F$="Test File "+STR$(I) : OPEN"O",1,F$
    PRINT#1,F$;" Created:",TIME$,DATE$ : CLOSE 1
    PRINT "Created: ";F$
  NEXT
END IF
PRINT"HIT ANY KEY TO RETURN TO FINDER"
DO : UNTIL LEN(INKEY$) : END
```

REMARK BUNDLE may also be set under the "Configure" menu item. Since most most stand-alone programs you create will require this to be set you may want to set this option and save that configuration.

Also see FINDERINFO, DEF OPEN, the sections under "RMaker™", "Using ICONS" and CREATOR for more information.

MACINTOSH APPENDIX

BUTTON function

FORMAT **BUTTON** (*button number*)

DEFINITION The BUTTON function returns the status of a specific BUTTON or SCROLL BUTTON (specified by the button number) from the current output window. Note that the number used for BUTTON and SCROLL BUTTON statements must never coincide.

USING THE BUTTON FUNCTION WITH *BUTTON*

The following values are returned for BUTTONs created with the BUTTON statement. The values returned are in the range of Ø to 2:

Ø	Not active (gray)
1	Active but not currently selected
2	Active and selected

USING THE BUTTON FUNCTION WITH *SCROLL BUTTON*

The following values are returned for Scroll bars created with the SCROLL BUTTON statement. The value returned depends on the action taken with that scroll bar. The range of values is determined in the SCROLL BUTTON statement by *min value* and *max value*. To determine what has occurred you will need to compare the old value with the new value:

Scroll Arrow clicked	New value =Old value plus or minus one (depending on which arrow was pressed)
Scroll Box moved	New value = relative position between arrows. min-max determine result. For instance; if the min value is 0 and the max value is 1000 and the box is positioned about 75% of the way over, the value returned will be about 750.
Scroll Bar clicked between the box and the arrow	New value=Old Value plus/minus *page up/down* value (see *page up/down* under SCROLL BUTTON).

EXAMPLE See the examples on next page, under the BUTTON statement and SCROLL BUTTON statement and also the example program SCROLL.BAS on the master disk for ways of opening text files and scrolling them in a window.

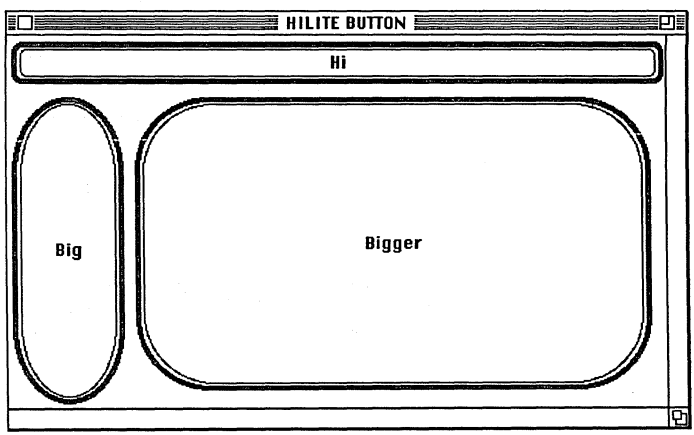
continued...

MACINTOSH APPENDIX

BUTTON function continued...

EXAMPLE

```
COORDINATE WINDOW : REM * THIS IS REQUIRED *
:
REM      Function to HiLight a Button
REM      (i.e. make a thick black box around it)
:
REM This function called by HiLiteButton function (BELOW)
LONG FN HiLite(x,y,x1,y1):REM  Pass rectangle size of button
  x=x-6 : x1=x1+6 : y=y-5 : y1=y1+5
  CALL PENSIZE(4,4) : r=(y1-y)>>1
  CALL SETRECT(t$,x,y,x1,y1)
  CALL FRAMEROUNDRECT(t$,r,r)
  CALL PENNORMAL
END FN
:
REM     Function to Open a new HiLited Button
REM     bn=Button number, bc=button condition
REM     T$=button title,  x,y = upper left corner
REM     x1, y1=Lower right corner of button
LONG FN HiliteButton(bn,bc,t$,x,y,x1,y1)
  BUTTON bn,bc,t$,(x,y)-(x1,y1),1
  FN HiLite(x,y,x1,y1)
END FN
:
WINDOW #1,"HILITE BUTTON"
FN HiliteButton(1,2,"Hi",10,10,480,30)
FN HiliteButton(2,2,"Big",10,50,80,270)
FN HiliteButton(3,2,"Bigger",100,50,470,260)
:
"LOOP" : BREAK ON
GOTO "LOOP"
```



REMARK Also see SCROLL BUTTON, BUTTON and DIALOG for more information about using buttons and scroll buttons. Also see the section in this appendix "WINDOWS" for more information about scrolling

MACINTOSH APPENDIX

BUTTON statement

FORMAT

BUTTON *[#] button number, condition [,title$, (x1,y1)-(x2,y2) [, type]]*
BUTTON CLOSE *[#] button number*

DEFINITION

ZBasic allows you to position buttons and other controls within a window for making "Mac" type programs. You will need to assign a number to each Button for identification purposes. Be sure that the SCROLL BUTTON and BUTTON statements do not use the same number.

The BUTTON statement displays a button in the output window at the position specified by x1,y1 in the type described in the BUTTON statement. The BUTTON CLOSE statement removes the button from the output window.

button number — The number used to identify a specific button. Numbers are integer and must be one or greater with a maximum of 32,767. This number must be unique and not coincide with SCROLL BUTTON numbers.

condition

Ø Button is not active (gray)
1 Button is active but not selected
2 Button is active and selected.

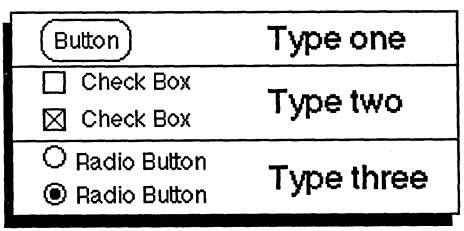
A button will remain active until:
- Another button statement is executed
- The window is closed
- That button is closed with the BUTTON CLOSE statement

*title$* — The string or text identifier used for the button. The title$ will be centered in type 1, and right-justified for types 2 and 3.

(x1,y1)-(x2,y2) — Where to position the button in the output window. (x1,y1) is the upper-left corner and (x2,y2) is the lower-right corner. Uses the ZBasic coordinates of 1024 x 768 or those defined by the COORDINATE statement. Values of X and Y default to Ø. Use COORDINATE WINDOW for pixel coordinates.

type

There are three types of buttons:
1 Simple box-type button with text in the center
2 Check box with text to the right
3 Radio button with text to the right



continued...

MACINTOSH APPENDIX

BUTTON statement continued

Buttons are a very easy way of getting input from a user (at least as far as the user is concerned).

To determine if a button has been pressed, use DIALOG ON and get the button event from the DIALOG(0) function.

EXAMPLE

```
WINDOW 1, "BUTTON WINDOW", (0,40)-(512,345),1
COORDINATE WINDOW: TEXT ,12
X1=20:Y1=30
:
FOR H=0 TO 2:Y1=30
  FOR X =1 TO 5
    Y1=Y1+20:Y2=Y1+15:X2=X1+40:A$=STR$(H*5+X)
    BUTTON H*5 + X, RND(3), A$, (X1, Y1)-(X2,Y2), H+1
  NEXT:X1=X1+100
NEXT: BUTTON 16, 1,"QUIT", (20,200)-(100,230),1
FLUSHEVENTS:DIALOG ON: ON DIALOG GOSUB"Dialog Check"
PRINT@(2,2);"TYPE 1"TAB(17)"TYPE 2"TAB(31)"TYPE 3";
:
"Main Event Loop"
GOTO "Main Event Loop"
:
DIALOG OFF
:
"Dialog Check"
B = DIALOG(0):B1=DIALOG(1)
IF B= AND B1=16 THEN END: REM Push Quit button to stop
BB=BUTTON (B1): BUTTON B1,3- BB
PRINT @(50,10);:CLS PAGE:PRINT "Dialog(0)="B;
PRINT @(50,11); "Button#"B1;
PRINT @(50,12); "Condition was:";BB
IF BB=3 THEN PRINT@(50,13)"Disabled";
RETURN
```

REMARK

When buttons are used, automatic text scrolling is disabled. Also see DIALOG, COORDINATE WINDOW, BUTTON function and EDIT FIELD for more information.

Note: Buttons will not function if a window is not opened with a ZBasic WINDOW statement.

Important Note: Do not close a window during a button event. If the system does not find the window for a button event a system error will occur.

MACINTOSH APPENDIX

CALL statement

FORMAT **CALL** *variable& [(expression, [expression,...])]*

DEFINITION This enhancement to the standard ZBasic CALL is used to pass parameters to a machine language routine. The data is pushed onto the stack. The last value pushed onto the stack is the return address (LongInteger).

variable The routine at the address contained in variable is called with the optional parameter list. No strings or arrays are allowed.

expression The expression is pushed onto the stack as a LongInteger (4 byte) except word variables (var% 2 byte integer) which are pushed as words. Floating point variables are truncated and converted to long integer (±2 billion).

To push an address, use VARPTR (variable). This will push the LongInteger address of that variable.

Strings pass the address of the string with a leading one byte length followed by 0-255 characters. String variables or quoted strings are used only for string parameters (no string expressions).

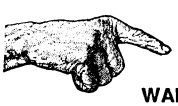
Pascal procedures may be called with this type. Use VARPTR for any VAR;Parameter.

EXAMPLE

```
REM Call a machine language program stored in A$
REM This is a fictional example . Do NOT use.
:
X&=VARPTR(A$)
CALL X& (10, 2000000, 300000, LINE "Continue")
"Continue"
program continues...
```

REMARK The routine being called MUST remove all parameters from the stack. i.e. If your routine requires 10 parameters it must pop 10 parameters off the stack. Otherwise a system error will occur ("Crash!#&@!").

Register conventions MUST be observed: A4, A5, A6 and A7 MUST be preserved. For compatibiliy with future versions do not use D4-D7. All others may be used.



WARNING: This command is for experienced, knowledgeable, 68000 machine language programmers.

MACINTOSH APPENDIX

CLEAR LPRINT statement

FORMAT **CLEAR LPRINT**

DEFINITION Forces the Macintosh Print Manager to print the current PAGE.

More than one CLEAR LPRINT may be used when printing a document.

EXAMPLE

```
WINDOW#1, "TEST"
BREAK ON
:
REM Example of Printing Text and Graphics
:
DEF PAGE: IF PRCANCEL THEN STOP
DEF LPRINT: IF PRCANCEL THEN STOP
:
COORDINATE 500,1000:     REM Set Printer Coordinate system
ROUTE 128:               REM Route stuff to the printer
:
FOR X=1 TO 5
  PEN,,,,RND(36)+1:      REM Change patterns of circle
  CIRCLE FILL RND(300)+100, RND(500)+250, RND (200)
NEXT
TEXT 3,24,9
PRINT%(250,500);"BYE";
CLEAR LPRINT:            REM Tell printer driver "All Done"
TEXT 2,12,0:             REM Set text back to normal
ROUTE 0:                 REM Route output back to the screen
END
```

REMARK A CLEAR LPRINT is automatically executed before a DEF LPRINT or DEF PAGE so that any text or graphics remaining in the buffer is released.

If this statement is omitted the last page (or none) of the printing will not occur until the program ends.

Also see WIDTH LPRINT-2, LPRINT, LPRINT%, LPRINT@, DEF LPRINT, DEF PAGE, ROUTE, COORDINATE, PRCANCEL, PRHANDLE, TEXT and PEN.

MACINTOSH APPENDIX

COMPILE command

FORMAT [L]COMPILE

DEFINITION Compiles the resident program and lists all compile time errors and memory segment partition data.

This command is used primarily to detect more of the errors in one pass (will only show one error per line).

LCOMPILE	Compiles the resident program and lists all the errors and segment partition information to the printer.
COMPILE	Compiles the resident program and lists all errors and segment partition information to the active output window.

EXAMPLE

```
ZBasic Ready

COMPILE
```

REMARK If compiling is accomplished without errors, you may type RUN and execution will be immediate (not RUN* or RUN+).

Note: COMPILE will cease if an Out-of-Memory error occurs. COMPILE is normally used from the Command Window but is also available under the Command menu.

Also see COMPILE in the main reference section of this manual.

SEGMENT PARTITION INFORMATION

COMPILE is also useful for determining SEGMENT information. Information returned:

Segment *s* ends In Stmt *xx* at LIne *nnnnn*: Size Is *mmmmm* bytes.

s	Segment number. Segments are contiguous and numbered 128, 129, 130 ...
xx	Last statement in last line of segment.
nnnnn	Line number where segment ends.
mmmmm	Length of SEGMENT in bytes.

See SEGMENT, SEGMENT RETURN and MEM (-1) for more information about controlling segments and memory managment.

MACINTOSH APPENDIX

CREATOR command

FORMAT **CREATOR***[=] "ffff"*

DEFINITION Sets four character "creator" information during program creation. For use during RUN*.

The "Creator" characters tell the FINDER what program "Created" certain files and will look for the creator file when one of it's files is double clicked.

A good example is when you double click a ZBasic Source Code file, the FINDER is nice enough to load ZBasic and ZBasic then loads your file. The FINDER knows that it is a ZBasic file because the Creator type is "ZBAS".

A ZBasic application is passed the information about what file it is to load (or print or whatever) through the FINDERINFO function.

EXAMPLE

```
ZBasic Ready

CREATOR="ZBAS"
```

REMARK The CREATOR may also be set from the "Configure" menu and from DEF OPEN. BUNDLE bit is also required.

This command must be executed from the Command Window.

If you are planning on marketing your product you may wish to contact Apple Computer to get your CREATOR and FILE types approved; (408) 973-4667.

Also see BUNDLE and FINDERINFO.

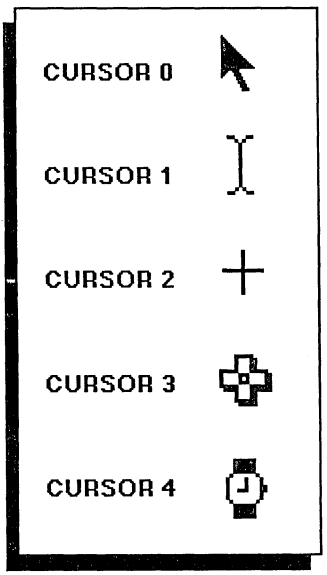
MACINTOSH APPENDIX

CURSOR statement

FORMAT **CURSOR***[=] expression*

DEFINITION The MOUSE controlled cursor appearance may be changed to a number of pre-defined system, or user defined resource cursor types using this statement.

Standard cursor numbers (in system file):



EXAMPLE1

```
WINDOW#1,"CURSOR"
"loop"
  INPUT"Cursor type";type
  IF type<0 or type>4 then STOP
  CURSOR=type
GOTO "loop"
```

continued...

MACINTOSH APPENDIX

CURSOR continued...

CREATE YOUR OWN MOUSE CURSORS

You may create customized mouse cursors using these toolbox calls:

CALL SETCURSOR *(integer array%)*
Creates a cursor image. The cursor consists of a 16 by 16 bit image. An integer variable array may contain the image for the cursor in the following format:

elements 0-15	cursor data
elements 16-31	mask data
element 32	vertical coordinate of the "HOT SPOT"
element 33	horizontal coordinate of the "HOT SPOT"

The "HOT SPOT " is the pointer position of the cursor. The normal arrow cursor "HOT SPOT" is the bit in the upper left corner of the cursor. This determines the point where the cursor is pointing to. To show the cursor, be sure to use CALL SHOWCURSOR.

See the "Toolbox" section of this appendix and *"Inside Macintosh"* for more information.

CALL	**INITCURSOR**	Reset cursor to the arrow shape.
CALL	**HIDECURSOR**	Hide cursor
CALL	**SHOWCURSOR**	Show cursor
CALL	**OBSCURECURSOR**	Hide cursor until mouse is moved
CHndl=	**FN GETCURSOR** (id)	Gets Handle to Resource Cursor ID
CALL	**SHIELDCURSOR** (Rect, Point)	Hide Cursor if in Rectangle area.

EXAMPLE2

```
CLS:BREAK ON
DIM A%(40)
DO
  CALL SETCURSOR(A%(0))
  X=X+1 : IF X=16 THEN X=0
  TRONX
  DELAY 20
  A%(0)=-1 : A%(15)=-1
  FOR I=1 TO 14
    A%(I)=(1<<X) OR 32769
  NEXT
  IF MOUSE(3) THEN PLOT TO MOUSE(1),MOUSE(2)
UNTIL LEN(INKEY$)
```

REMARK You may also add or change cursor types as resources with ResEdit or RMaker. See CURSOR in the toolbox section of this manual.

MACINTOSH APPENDIX

DEFDBL INT statement

FORMAT **DEFDBL INT** *alpha, alpha range -alpha range*

DEFINITION Defines selected variables as LongInteger variable type and variables .

This variable type offers an enhanced integer range of -2147,483,648 to +2,147,483,647 (versus -32,768 to 32,767 for regular integer variables.)

EXAMPLE

```
DEFDBL INT A-G
A=100000*2:?A

RUN

200000
```

REMARK LongIntegers are also set by using the "&" symbol. See "LongInteger" in this appendix and DEF var in the main reference section.

Also see DEFSTR LONG and DEFSTR WORD for using LongIntegers with MKI$, CVI, HEX$, UNS$, OCT$, BIN$, DEFSTR, DEFSNG, DEFDBL and DEFINT.

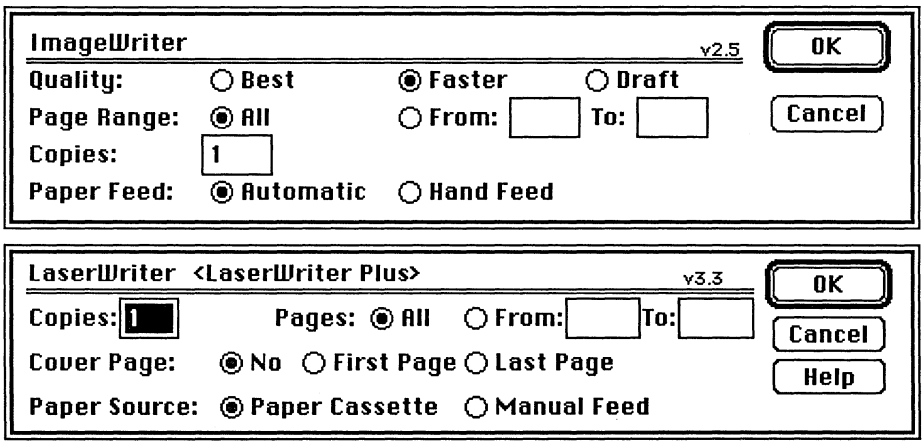
MACINTOSH APPENDIX

DEF LPRINT statement

FORMAT DEF LPRINT

DEFINITION Brings up the printer formatting window and initializes the Macintosh printer driver. Allows the user to change the print to draft, standard or high quality output. The default is the last one used. See PRHANDLE function for way of getting the parameters set by the user.

Be sure to use CLEAR LPRINT to clear out the last printer buffer page.



EXAMPLE

```
WINDOW#1, "DEF LPRINT":WIDTH LPRINT-2
DEF PAGE: REM Do printer set-up first!
DEF LPRINT
IF PRCANCEL THEN PRINT"Printing Aborted":STOP
LPRINT"HELLO!"
:
ROUTE 128: REM This routes screen output to the printer
CIRCLE FILL 200,200,200
PRINT"GOODBYE"
CLEAR LPRINT
ROUTE 0:   REM This routes output back to the screen.
END
```

REMARK To determine if the "Cancel" button has been pressed use the PRCANCEL function.

To finish the printing the current page use the CLEAR LPRINT statement. <COMMAND PERIOD> will abort printing in progress.

Also see LPRINT% and LPRINT@ for use with text output to printers and for faster printing use WIDTH LPRINT -2. See ROUTE 128 and the section in the front of this manual for details about using the Imagewriter and LaserWriter printers.

MACINTOSH APPENDIX

DEF MOUSE statement

FORMAT **DEF MOUSE [=] *n***

DEFINITION This is used to define ZBasic or MSBASIC type MOUSE syntax . If this statement is not used the default is ZBasic syntax (DEF MOUSE=0).

To use the MSBASIC™ MOUSE syntax use DEFMOUSE=-1.

The values of n are:

n = Ø	ZBasic compatible mouse syntax. Same as the main reference manual (default).
n <> Ø	Defines the MOUSE statements and functions to use the MSBASIC™ type MOUSE syntax (see MOUSE in this appendix).

EXAMPLE

```
WINDOW#1:COORDINATE WINDOW
DEF MOUSE=1: REM All subsequent MOUSE commands will be MSBASIC syntax
program continues...
```

NOTE: If MOUSE ON is used DEFMOUSE MUST be used or a crash will result!

REMARK ***MSBASIC MOUSE versus ZBASIC MOUSE STATEMENTS***

While we have attempted to make the MOUSE commands as completely compatible as possible, there are certain circumstances that will cause different results.

ZBasic checks events only at the beginning of each line physically between a MOUSE ON and MOUSE OFF statement. An interpreter will check for mouse events even when a GOSUB is made between MOUSE ON and a MOUSE OFF statements. ZBasic will not.

ZBasic does not check for events at each statement in a multiple statement line ONLY AT THE BEGINNING OF EACH LINE.

The event checking does not take effect until the next line following the MOUSE ON statement. Therefore the following program line would never check for a MOUSE event:

```
MOUSE ON: ON MOUSE GOSUB 1000: MOUSE OFF
```

In addition, we recommend that you put multiple Event ON structures on the same line for efficiency. Like this:

```
MOUSE ON: DIALOG ON: BREAK ON
```

As with all other event types, we recommend that you execute a MOUSE OFF statement whenever time critical calculations or functions are being performed. ANY events left active will slow down program execution significantly (not turned OFF).

Also see COORDINATE WINDOW in the main reference section and "Converting MSBASIC programs" in this appendix.

MACINTOSH APPENDIX

DEF OPEN statement

FORMAT **DEF OPEN** *[=] "ffffcccc"*

DEFINITION *ffff* sets the four character "FILE" type of files that are opened in subsequent program lines. File types are used with the Macintosh™ to inform the system of several things.

For instance, if you wanted to create a file that could be loaded into MacWrite™ or Microsoft Word™, you would give it a filetype of "TEXT" (uppercase mandatory).

cccc sets the "CREATOR" type to be used. This is primarily for files your application creates. This informs the Macintosh system to use your application to execute that file.

ZBasic source code files have a CREATOR type of "ZBAS". If someone clicks a file with a ZBAS "Creator type", Finder™ will load ZBasic so that it can execute or load that file. Other examples of "TYPE" and "CREATOR":

APPLICATION	TYPE	CREATOR
MacPaint	PNTG	MPNT
MacDraw	DRWG	MDRW
Font DA/Mover	DFIL	DMOV
System	ZSYS	MACS
Finder	FNDR	MACS
ZBasic™	APPL	ZBAS

The default data FILE TYPE is "ZDAT". To see examples of file types see DIR.

EXAMPLE1

```
DEF OPEN "DATATHNG"
OPEN O,1,"SAMPLE DATA"
FOR I = 1 TO 100
  PRINT #1,I
NEXT
CLOSE 1
```

This program above creates a data file called "SAMPLE DATA" with the file type of DATA and a creator of THNG.

In command mode type CREATOR "THNG" and then type in this program and use RUN*. This identifies this application program as "THNG". It doesn't matter what the actual filename is (but for this example call it "FRED") when you save it with RUN*.

REMARK When double clicking the data file "SAMPLE DATA" created in the first program above, the Macintosh Finder™ sees that the creator of this file was "THNG" and will load "FRED" because it's "creator" was defined as "THNG". Use RMAKER to define Icons, FREFS and version data.

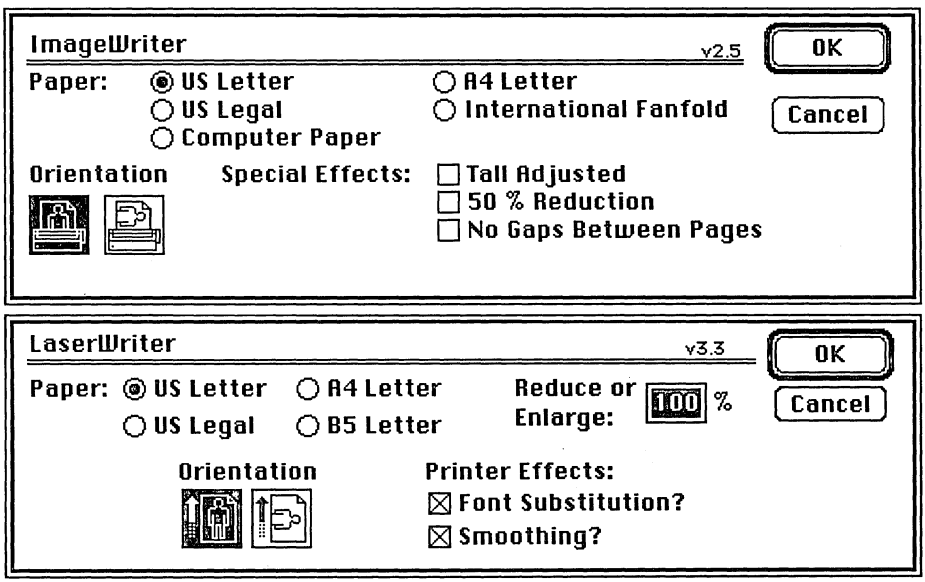
NOTE: If you are creating a program that is going to be marketed, you may want your file type or creator type approved by Apple Computer, Inc. (408) 973-4667

MACINTOSH APPENDIX

DEF PAGE statement

FORMAT DEF PAGE

DEFINITION Brings up the page formatting dialog box. Allows the user to set parameters for printing. To determine what the user chose see PRHANDLE.



EXAMPLE

```
WINDOW#1,"DEF PAGE":WIDTH LPRINT-2
DEF PAGE: IF PRCANCEL THEN PRINT"CANCEL PRESSED":STOP
DEF LPRINT: IF PRCANCEL THEN PRINT"CANCEL PRESSED":STOP
FOR X=1 TO 10
  LPRINT@(RND(30,30))"HELLO"
NEXT X
CLEAR LPRINT
END
```

REMARK See "Using the LaserWriter and Imagewriter" in the front of this appendix for more information. Also be sure to see DEF LPRINT, ROUTE 128, PRCANCEL, PRHANDLE and WIDTH LPRINT -2.

Be sure to use CLEAR LPRINT to print the current and last page of a document.

MACINTOSH APPENDIX

DEFSTR LONG statement
DEFSTR WORD statement

FORMAT **DEFSTR LONG**
DEFSTR WORD

DEFINITION Since the Macintosh version of ZBasic supports both regular and Long Integer variables these statements are used to define the current type:

DEFSTR LONG	Four byte	LongInteger	range ±2 billion)
DEFSTR WORD	Two byte	Regular integer	range ±32,767)

These statements affect the following ZBasic commands:

HEX$	UNS$
OCT$	BIN$
MKI$	

LongIntegers require four bytes and regular integers require two bytes.

EXAMPLE

```
DEFSTR WORD
X$=MKI$(23021)
PRINT LEN(X$)
:
DEFSTR LONG
X$=MKI$(2032102)
PRINT LEN(X$)
END

RUN

2
4
```

REMARK Also see "Defining Variable Types" in the main reference section and the sections on "LongInteger" and DEFDBL INT in this appendix.

MACINTOSH APPENDIX

DIALOG function

FORMAT **DIALOG** *(expression)*

DEFINITION Returns the next event in the buffer (queue). After reading, the event it is removed from the buffer (First-in, First-out buffer [FIFO]). If no events are in the buffer a zero is returned.

To determine what DIALOG event has occurred use *event%* =DIALOG(Ø). To determine more detailed information about that event use; DIALOG(*event%*):

DIALOG (Ø) Using DIALOG(Ø) tells you the type of event that has occurred. This lets you branch to that part of your program.

For instance if DIALOG(Ø)= zero there are no events in the queue. If DIALOG(Ø)=4, a close-window box has been clicked and you should put away the window specified by DIALOG(4).

DIALOG TYPES	DIALOG(Ø) RETURNS	
NONE	Ø	No dialog event has occurred.
BUTTON	1	A BUTTON event has occurred. DIALOG(1) returns the number of the button that was pressed.
EDIT FIELD	2	Movement from one EDIT FIELD to another has occurred. Use DIALOG(2) to find out what EDIT FIELD was selected. There must be more than one edit field in the window for this function to work.
Inactive WINDOW	3	An INACTIVE WINDOW has been clicked. To find out what window number was clicked use DIALOG(3). You will probably want to activate that window. See the chapter "WINDOWS" and the WINDOW statement and function for more information.
CLOSE BOX	4	The CLOSE WINDOW BOX has been clicked. The window number to close is returned in DIALOG(4). See the chapter "WINDOWS" and the WINDOW CLOSE statement for more information.
WIndow UPDATE	5	A portion of the current ACTIVE WINDOW has been ERASED by a dialog window, Desk Accesory or something else and will need to be refreshed. The window number that needs to be refreshed is returned in DIALOG(5). Except for menu action, you must re-draw the window whenever parts of it are erased. ZBasic does not re-draw the window automatically. See the chapter in this appendix "WINDOWS" for ways of doing this. Also see WINDOW PICTURE.

continued...

MACINTOSH APPENDIX

DIALOG function continued

DIALOG TYPES		DIALOG(Ø) RETURNS
		---------- Only when EDIT FIELD is active ----------
RETURN KEY	6	The RETURN key was pressed in an active window that is using EDIT FIELDS. You may want to accept this as a message to accept the input as "OK" or as the default (highlighted).
TAB KEY	7	The TAB key was pressed. This may signify a move to the next field or button when EDIT FIELD is used. Also see events 10-16 below.
ZOOM-IN	8	The ZOOM-IN box has been clicked by the user. Get the WINDOW number from DIALOG(8) (window type 9 only).
ZOOM-OUT	9	The ZOOM-OUT box has been clicked. Get the WINDOW number from DIALOG(9) (window type 9 only).
		---------- Only when EDIT FIELD is active ----------
SHIFT-TAB	10	SHIFT-TAB has been pressed while using EDIT FIELD. This is often used to signify a BACK-UP to the last EDIT FIELD. The current EDIT FIELD number is returned in DIALOG(10). Also see LEFT-ARROW.
CLEAR KEY	11	The CLEAR key has been pressed while using EDIT FIELD. This often signifies an ERASE FIELD. The current EDIT FIELD number is returned in DIALOG(11).
LEFT-ARROW	12	The LEFT-ARROW cursor was pressed while using EDIT FIELD. This may signify a BACK-UP to the previous EDIT FIELD directive. The current EDIT FIELD number is returned in DIALOG(12).
RIGHT-ARROW	13	The RIGHT-ARROW cursor was pressed while using EDIT FIELD. This signifies a MOVE-AHEAD to the next EDIT FIELD directive. The current EDIT FIELD number is returned in DIALOG(13).
UP-ARROW	14	The UP-ARROW cursor was pressed while using the EDIT FIELD. This often signifies a MOVE-UP to the EDIT FIELD above directive. The current EDIT FIELD number is returned in DIALOG(14).
DOWN-ARROW	15	The DOWN-ARROW cursor was pressed while using EDIT FIELD. This often signifies a MOVE-DOWN to the EDIT FIELD below directive. The current EDIT FIELD number is returned in DIALOG(15).
		---------- Only when EDIT FIELD is NOT active ----------
EVENT INKEY$	16	A KEY was pressed. The key pressed is returned in DIALOG(16) as the ASCII value of the key. Since INKEY$ function conflicts with event trapping, this is the best way to trap keys when doing: DIALOG ON, BREAK ON, MENU ON, TIMER ON and any other events (TRON conflicts with DIALOG(16)).

continued...

MACINTOSH APPENDIX

DIALOG function continued

The following functions are paired with the DIALOG (Ø) function. If DIALOG(0) returns a one then check DIALOG(1). If DIALOG(Ø) returns a five, then check DIALOG(5). The lines between the types signify like functions.

DIALOG FUNCTION	RETURNS
DIALOG(1)	**NUMBER of BUTTON CLICKED.** Use to determine which button has been clicked.
DIALOG(2)	**NUMBER of EDIT FIELD CLICKED.** Normally used to "check" the contents of the previous entry. Function requires more than one EDIT FIELD in the active window in order to function.
DIALOG(3)	**NUMBER OF AN INACTIVE WINDOW THAT HAS BEEN CLICKED.** If a window behind or to the side of the active window is clicked it usually means the user wants to activate this window.
DIALOG(4)	**WINDOW NUMBER WHOSE CLOSE BOX WAS CLICKED.** The Close box is the box in the upper left hand corner of window types 1 and 5 and when clicked usually means the user wants to "exit" the program or close that window.
DIALOG(5)	**NUMBER OF WINDOW THAT HAD A PORTION ERASED.** When a window erase event occurs you will need to refresh (re-draw) the contents of that window.
	Only when EDIT FIELD is active
DIALOG(6)	**EDIT FIELD number** where the **RETURN** key was pressed.
DIALOG(7)	**EDIT FIELD number** where the **TAB** key was pressed.
	Only with 128K ROM
DIALOG(8)	**Window number** whose **ZOOM-IN** box was clicked.
DIALOG(9)	**Window number** whose **ZOOM-OUT** box was clicked.

continued...

MACINTOSH APPENDIX

DIALOG function continued

FUNCTION	RETURNS
---------- Only when EDIT FIELD is active ----------	
DIALOG(10)	**EDIT FIELD number** where the **SHIFT-TAB** keys were pressed.
DIALOG(11)	**EDIT FIELD number** where the **CLEAR** key was pressed.
DIALOG(12)	**EDIT FIELD number** where the **LEFT-ARROW** key was pressed.
DIALOG(13)	**EDIT FIELD number** where the **RIGHT-ARROW** key was pressed.
DIALOG(14)	**EDIT FIELD number** where the **UP-ARROW** key was pressed.
DIALOG(15)	**EDIT FIELD number** where the **DOWN-ARROW** key was pressed.
---------- Only when EDIT FIELD is NOT active ----------	
DIALOG(16)	**Returns the ASCII code of the last key pressed during event trapping.** If 65 is returned than the "A" was pressed (CHR$(65)=A). Use this instead of INKEY$ when event trapping is enabled since INKEY$ will not function properly during event trapping.

THE EVENT QUEUE (BUFFER)

Up to 64 Events are stored in the DIALOG function queue (a FIFO holding buffer).

For instance, if you click a BUTTON 4 times DIALOG (Ø) would contain 4 levels of the number 1. You would have to execute DIALOG(Ø) five times before DIALOG(Ø) would equal zero.

Sometimes you may want to "flush out" old events. For example when you open a window there is usually a WINDOW refresh event that needs to flushed or ignored. Use FLUSHEVENTS.

Also see section on "WINDOWS" in this appendix and EDIT FIELD, EDIT$, WINDOW statements and functions, BREAK ON and "Writing a Macintosh Program" in this appendix.

See DIALOG Statement for example program.

MACINTOSH APPENDIX

DIALOG statement

FORMAT

DIALOG ON
DIALOG OFF
DIALOG STOP
ON DIALOG GOSUB *line or label*

DEFINITION

DIALOG ON	Enables event trapping for EDIT FIELD, BUTTON and WINDOW activity. To determine which event has taken place use DIALOG(Ø). If DIALOG(Ø) =Ø then no event has taken place. The event is checked at the beginning of each line, not between statements.
DIALOG OFF	Discontinue checking and storing DIALOG events.
DIALOG STOP	Temporarily stops scanning the event buffer until a DIALOG ON statement is encountered. Events will be stored until the events are checked again.

Event trapping will significantly slow execution speed. When high speed is an important factor be sure to execute a DIALOG OFF statement.

EXAMPLE1

```
REM ZBasic Dialog Example
REM ©MacTutor 1987
REM By Dave Kelly (Thanks for letting us use this Dave! mg)
WINDOW OFF
COORDINATE WINDOW:REM Set window to pixel coordinate system
False=0:True=NOT False
IF PEEK(&28E) AND 128 THEN Wtype=1 ELSE Wtype=9:REM Which ROM?
:
MENU 1,0,1,"File"
MENU 1,1,1,"Quit"
WINDOW 1,"Window 1",(10,50)-(250,200),Wtype
TEXT 4,9,0,0
BUTTON 1,1,"Button 1",(20,20)-(100,50)
BUTTON 2,1,"Button 2",(20,60)-(100,90)
WINDOW 2,"Window 2",(275,50)-(500,200),Wtype
TEXT 4,9,0,0
EDIT FIELD 1,"",(10,10)-(100,35),1,1
EDIT FIELD 2,"",(10,40)-(100,65),1,1
WINDOW 3,"Dialog Event (Window #3)",(10,250)-(500,340),28
TEXT 4,9,0,0
ON DIALOG   GOSUB "DialogEvent"
ON BREAK    GOSUB "BreakEvent"
ON MENU GOSUB "MenuEvent"
DIALOG ON:BREAK ON:MENU ON
:
"Mainloop":
GOTO "Mainloop"
:
DIALOG STOP : BREAK STOP : MENU STOP
```

MACINTOSH APPENDIX

```
"MenuEvent"
Menunumber=MENU(0):Itemnumber=MENU(1)
IF Menunumber=1 AND Itemnumber=1 THEN END
RETURN
"BreakEvent"
STOP
"DialogEvent"
D = DIALOG(0):REM check to see what event occured
"DEvent"
Currentwindow = WINDOW(0)
Windowselection = WINDOW(1)
DIALOG OFF : BREAK OFF: MENU OFF
WINDOW OUTPUT 3
IF D = 1 GOSUB "Buttonevent"
IF D = 2 GOSUB "EditEvent"
IF D = 3 GOSUB "InactiveWindow"
IF D = 4 GOSUB "Closebox"
IF D = 5 GOSUB "Refresh"
IF D = 6 GOSUB "Returnkey"
IF D = 7 GOSUB "Tabkey"
IF D = 8 GOSUB "Zoomin"
IF D = 9 GOSUB "Zoomout"
IF D =10 GOSUB "Shifttab"
IF D =11 GOSUB "Clearkey"
IF D =12 GOSUB "LeftArrow"
IF D =13 GOSUB "RightArrow"
IF D =14 GOSUB "UpArrow"
IF D =15 GOSUB "DownArrow"
IF D =16 GOSUB "Keypress"
PRINT @(50,3) "DIALOG(0) :        ";D
PRINT @(50,4) "Active Window #";Currentwindow
PRINT @(50,5) "Output Window #";Windowselection
WINDOW OUTPUT Outwindow:WINDOW Windowselection
RETURN
"Buttonevent"
Buttonclicked=DIALOG(1)
Bstatus=BUTTON(Buttonclicked):BUTTON Buttonclicked,3-Bstatus
B=B+1:PRINT@(1,1)   "Button clicked :   ";Buttonclicked;B
RETURN
"EditEvent":
EditField=DIALOG(2)
PRINT@(1,2)   "Edit Field :       ";EditField
RETURN
"InactiveWindow"
Windowselection=DIALOG(3)
PRINT@(1,3)   "Inactive Window :  ";Windowselection
RETURN
"Closebox":
ClosedWindow=DIALOG(4)
IF ClosedWindow=3 THEN END
PRINT@(1,4)   "Closed Window :    ";ClosedWindow
RETURN
```

MACINTOSH APPENDIX

```
"Refresh":
ErasedWindow=DIALOG(5)
REM WINDOW OUTPUT DIALOG(5) : CLS : WINDOW OUTPUT 3
PRINT @(1,5)  "Erased Window :     ";ErasedWindow
RETURN
"Returnkey":
Returnpress=DIALOG(6)
PRINT @(25,1) "Return press :      ";Returnpress
RETURN
"Tabkey":
Tabpress=DIALOG(7)
PRINT @(25,2) "Tab press :         ";Tabpress
RETURN
"Zoomin": REM NEW ROMS ONLY
Zin=DIALOG(8)
WINDOW Zin
PRINT@(1,1) "Thank you for zooming in window";Zin
PRINT @(25,3) "Zoom in window :    ";Zin
RETURN
"Zoomout":
Zout=DIALOG(9)
WINDOW Zout
PRINT@(1,1)"Thank you for zooming out window";Zout
PRINT @(25,4) "Zoom out window :   ";Zout
RETURN
"Shifttab":
CurrentEdit=DIALOG(10)
PRINT @(25,5) "Current Edit Field:";CurrentEdit
RETURN
"Clearkey":
CurrentEdit=DIALOG(11)
PRINT @(25,5) "Current Edit Field:";CurrentEdit
RETURN
"LeftArrow":
CurrentEdit=DIALOG(12)
PRINT @(25,5) "Current Edit Field:";CurrentEdit
RETURN
"RightArrow":
CurrentEdit=DIALOG(13)
PRINT @(25,5) "Current Edit Field:";CurrentEdit
RETURN
"UpArrow":
CurrentEdit=DIALOG(14)
PRINT @(25,5) "Current Edit Field:";CurrentEdit
RETURN
"DownArrow":
CurrentEdit=DIALOG(15)
PRINT @(25,5) "Current Edit Field:";CurrentEdit
RETURN
"Keypress":
ASCIIkey=DIALOG(16)
PRINT @(50,1) "ASCII key pressed :";ASCIIkey;"  "
PRINT @(75,1) " ":PRINT @(75,1) CHR$(ASCIIkey)
RETURN
END
```

continued...

MACINTOSH APPENDIX

DIALOG statement continued...

EXAMPLE2

```
REM +-------------------------------------------------------------+
REM +  Event Handling Template Example ZBasic 4.0 or Greater    +
rem +                  By Andrew Gariepy,  5/87     (ON MASTER DISK)+
REM +-------------------------------------------------------------+
WINDOW OFF  : COORDINATE WINDOW  : WIDTH -2   : DEF MOUSE 1
WINDOW #1,"Example",,9 : APPLE MENU "Help ^1" : MENU 1,0,1,"File"
MENU 1,1,1,"Open/O;Save/S;(-;Page Setup;(-;Print/P;(-;Quit/Q"
EDIT MENU 2 : MENU 2,7,1,"(-;Bold<B;Italic<I;Outline<O;Shadow<S"
SCROLL BUTTON 1,0,0,100,10,,1 :  SCROLL BUTTON 2,0,0,100,10,,2
ON DIALOG   GOSUB "Dialog": REM Vector to DIALOG routine
ON MENU     GOSUB "Menu"  : REM Vector to MENU   routine
ON BREAK    GOSUB "Break" : REM Vector to BREAK  routine
ON TIMER(5) GOSUB "Timer" : REM Vector to TIMER  routine (Seconds)
ON MOUSE    GOSUB "Mouse" : REM Vector to MOUSE  routine (DEF MOUSE 1)
REM Main Event Loop
FLUSHEVENTS: DIALOG ON: MENU ON: TIMER ON: MOUSE ON: BREAK ON
DO        : REM Just a loop While waiting for events
UNTIL 0   : REM DO FOREVER.....
DIALOG OFF: MENU OFF: TIMER OFF : MOUSE OFF: BREAK OFF
REM Dialog Event Handler
"Dialog" : D0=DIALOG(0) : DN=DIALOG(D0) : REM Get Event & Value
ON D0    GOTO "Button","Edit","Activate","Close","UpDate","Return"
ON D0-6  GOTO "Tab Key","Zoom-In","Zoom-Out","ShiftTab","ClearKey"
ON D0-11 GOTO "Left Key","RightKey","Up Key","Down Key","OtherKey"
PRINT "Un-Defined Dialog #";D0;"With a Value=";DN : RETURN
"Activate":REM ---------- Window #DN clicked -----------
WINDOW #DN : RETURN : REM *** Typical Activate Event ***
"Close"   :REM ---- Close Box of Window #DN Clicked ----
WINDOW CLOSE #DN : RETURN : REM Typical Window Close Event
"UpDate" : REM --- Part of Window #DN Needs Updating ---
"Button" : REM ------ Button Event for Button #DN ------
"Edit"   : REM -------- Edit Field #DN Selected --------
"Return" : REM ---- Return/Enter in Edit Field #DN -----
"Tab Key": REM --- Tab Key Pressed in Edit Field #DN ----
"Zoom-In": REM ----- Zoom in Event for Window #DN ------
"Zoom-Out":REM ---- Zoom Out Event for Window #DN ------
"ShiftTab":REM ------ Shift-Tab in Edit Field #DN ------
"ClearKey":REM ------ Clear Key in Edit Field #DN ------
"Left Key":REM ------ Left Key in Edit Field #DN -------
"RightKey":REM ----- Right Key in Edit Field #DN -------
"Up Key"  :REM ------- Up Key in Edit Field #DN --------
"Down Key":REM ------ Down Key in Edit Field #DN -------
"OtherKey":REM ------- Other Key, Key$=CHR$(DN) --------
PRINT "Event #";D0;"with value of";DN;"Not Handled!":RETURN
REM Menu Event Handler
"Menu" :    M0=MENU(0) : M1=MENU(1) : REM Get Menu and Item
IF M0=255   THEN  PRINT "Apple Menu Help Selected"
ON M0 GOSUB "File Menu","Edit Menu" : REM Menu Title Vectors
MENU : RETURN : REM *** Un-HiLight Menu Title & Return ****
"File Menu"   : REM *** All file Menu items come here *****
ON M1 GOTO "Open","Save","Z","Setup","Z","Print","Z","Break"
"Open"  : PRINT "Open File Menu Selected" : RETURN
"Save"  : PRINT "Save File Menu Selected" : RETURN
"Setup" : DEF PAGE   : RETURN
"Print" : DEF LPRINT : LCOPY : RETURN
"Z"     : RETURN : REM Just Return Entry
"Edit Menu"  : RETURN : REM ** Your Edit Menu items **
REM Break,Timer & Mouse Event Handlers
"Break": BEEP : BEEP : BEEP : BEEP : STOP : REM Debug
"Timer": PRINT "Timer, Active Window=";WINDOW(0) : RETURN
"Mouse": M0=MOUSE(0) : PRINT "Mouse Event" : RETURN
```

MACINTOSH APPENDIX

DIR command

FORMAT *[L]* **DIR** *[pathname]*

DEFINITION This version offers enhancements to the standard disk directory command. To pause the directory press <SPACE>. Any other key will continue the listing. <COMMAND PERIOD> to abort the listing.

DIR	Lists the folders and files in the current directory
LDIR	Lists directory to the printer
DIR Harddisk	Lists the directory of the Root volume called Harddisk.
LDIR Harddisk:Examples	Lists directory of the folder "Examples" in the root directory called "Mydisk", to the printer.
DIR::	BACKS-OUT of a folder to the previous folder level.

EXAMPLE

```
ZBasic Ready
DIR

**ZBASIC(0)             HFS(1)              220 k Free (2)

(3)                     (4)    (5)    (6)       (7)       (8)
Desktop                 FNDR   ERIK   16457     16896     I
System                  ZSYS   MACS   228948    230912
BAS Examples            Folder
ZBasic                  APPL   ZBAS   85214     86016
ZEXAMPLES               Folder
```

(0) The Current disk volume or folder name
(1) Type of System: HFS or MFS
(2) Amount of free space on disk
(3) File or Folder name
(4) Type. ZBasic application types are APPL. Help file types are ZHLP. Data files are ZDAT. ASCII files are TEXT.
(5) Creator. The program that created the file. For instance, when you click a paint document it loads Macpaint automatically. The ZBasic creator name is ZBAS.
(6) Actual end of file (in bytes)
(7) Amount of disk space allocated in bytes.
(8) I=invisible file. L=locked file. P=protected file.

REMARK **NOTE**: It is important to note that if you want the directory of another folder other than the current one, you will need to give the complete pathname starting with the root volume or you will need to BACK-OUT to the main volume using " DIR :: ".

Folder and filenames MUST be spelled correctly in order for this command to work. If the pathname is not valid no directory will be displayed.

Also see FILE$, KILL, DEF OPEN, EJECT, CREATOR, and BUNDLE and in this appendix and DIR in the main reference section.

MACINTOSH APPENDIX

EDIT FIELD statement

FORMAT **EDIT FIELD** *field number[,simplestring[, (x1,y1)-[(x2,y2)[,[type][,format]]]]]*
EDIT FIELD CLOSE *field number*

DEFINITION EDIT FIELD is used to allow a user to enter text at a specific region in the current output window. This statement is handy for creating easy Macintosh type edit input for the user.

field number — An integer number. This number may be an expression. It identifies the edit field within the current window.

simplestring — A string that will appear in the edit field. " " may be used to specify no string field.

(x1,y1)-(x2,y2) — Specifies the rectangle in which the editing is to occur. x1,y1 specifies the upper left corner, x2,y2 specifies the lower right.

type — Tells ZBasic how to format the edit field:

1	Enclosed in a Box. RETURN key ignored. See DIALOG(6).
2	Enclosed in a Box. "RETURN" allowed.
3	No Box. RETURN ignored. See DIALOG(6)
4	No Box. RETURN allowed.

format — Tells ZBasic how to format the text in the edit field:

1	Left Justify
2	Center
3	Right Justify

EDIT FIELDS are specific to the window being used. See EDIT$ function to see how to retrieve the data typed into the edit field.

Use EDIT MENU with this statement to allow the user to do pasting, cutting and copying. Also see TEHANDLE for ways of manipulating text in EDIT FIELDS larger than 255 characters.

EDIT FIELD CLOSE *field number* Closes the edit field specified by field number.

EXAMPLE

```
COORDINATE WINDOW:DEFMOUSE=1
WINDOW 1, "EDIT FIELD", (41,51)-(469,306),1
TEXT 0,12,0
:
BUTTON 1,1,"QUIT",(278,173)-(376,197),1
EDIT MENU 2:REM Allow CUT and PASTE
EDIT FIELD 1,"Edit Field 1",(39,42)-(205,58),1
EDIT FIELD 2,"Edit Field 2",(210,42)-(376,58),1
EDIT FIELD 3,"Edit Field 3",(39,74)-(205,90),1
EDIT FIELD 4,"Edit Field 4",(210,74)-(376,90),1
EDIT FIELD 5,"Edit Field 5",(39,106)-(205,122),1
EDIT FIELD 6,"Edit Field 6",(210,106)-(376,122),1
:
FLUSHEVENTS: DIALOG ON:BREAK ON
ON DIALOG GOSUB "Dialog Events"
```

continued...

MACINTOSH APPENDIX

```
"Main Event Loop"
GOTO "Main Event Loop"
:
DIALOG OFF:BREAK OFF
"Dialog Events"
Event=DIALOG(0):What=DIALOG(Event)
IF Event=1 THEN "End":     REM Push button=End
SELECT Event:              REM Check DIALOG 6,7,10,11,12,13,14,15
  CASE 6,7,13:             REM Move to next field
    LONG IF What<6
      EDIT FIELD# What+1
    XELSE:                 REM Move to First if at Last
      EDIT FIELD#1
    END IF
  CASE 10,12:              REM Move back to last field
    LONG IF What>1
      EDIT FIELD# What-1
    XELSE
      EDIT FIELD#6:        REM Move to Last if at First
    END IF
  CASE 14:                 REM Move UP
    SELECT What
      CASE 3,4,5,6
        EDIT FIELD#What-2
      CASE 1,2
        EDIT FIELD#What+4
    END SELECT
  CASE 15:                 REM Move DOWN
     SELECT What
       CASE 1,2,3,4
         EDIT FIELD#What+2
       CASE 5,6
         EDIT FIELD#What-4
     END SELECT
  CASE ELSE
    EDIT FIELD#What
END SELECT
RETURN
:
"End"
BEEP:BEEP:CLS:PRINT"FINAL Contents of Edit Fields:"
FOR X=1 TO 6
  PRINT TAB(25); EDIT$(X):EDIT FIELD CLOSE#X
NEXT: DELAY 5000: END
```

REMARK **Note**: The EDIT FIELD does not stop and wait for the user, you must trap EDIT FIELD events using DIALOG. DIALOG functions 2, 6,7,10,11,12,13,14 and 15 return EDIT FIELD event and key information. Use EDIT MENU to enable CUT, COPY and PASTE.

Be sure to see the DIALOG statement and function and EDIT$ for more information. See TEHANDLE for using edit fields greater than 255 characters.

Note: EDIT FIELD events will not function in the ZBasic default window. You will have to open your own window before the event is detected properly.

MACINTOSH APPENDIX

EDIT MENU statement

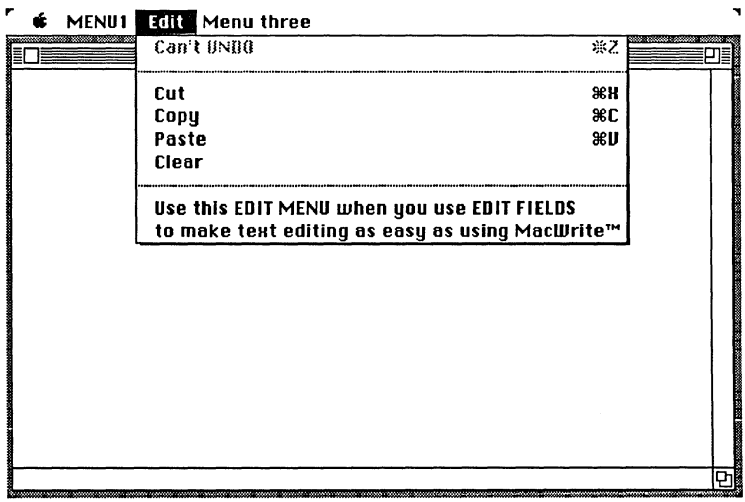
FORMAT **EDIT MENU** *menu number*

DEFINITION This statement is used to bring up the standard Macintosh "EDIT" menu items specifically for use with EDIT FIELDS.

This allows the user to CUT, COPY, PASTE and CLEAR text editing activity automatically and saves the programmer the hassle of having to write special routines to do this.

The EDIT menu number should normally be assigned number two (File is normally menu number one).

EXAMPLE See example under EDIT FIELD statement.



REMARK EDIT MENU Ø disables the menu item.

Note that UNDO is not supported directly. If you add items to this menu or enable UNDO, item 1 it will return the item selected other than CUT, COPY PASTE and CLEAR.

MACINTOSH APPENDIX

EDIT$ function

FORMAT EDIT$ *(number)*

DEFINITION Returns the current text in the EDIT FIELD specified by *number*.
Use EDIT FIELD to set up the input field size and default input.

EXAMPLE See example under EDIT FIELD statement.

REMARK Closing the edit field will delete the contents.

See TEHANDLE for example of using an EDIT FIELD with up to 32000 characters.

MACINTOSH APPENDIX

EJECT statement

FORMAT **EJECT** {[-]1 or [-]2}

DEFINITION Ejects the diskette in the specified drive:

EJECT number	RESULT
EJECT 1	Ejects the diskette in the internal drive. Disk volume is retained so programs accessing that volume number will result in the system asking for you to insert that diskette (no fun).
EJECT -1	Ejects the diskette in the internal drive. Disk volume is not retained; a file-not-found error will result if you attempt to access that volume again. Note that the volume number may change if the diskette is inserted again after having been ejected in this way.
EJECT 2 and -2	Same as above except for external drive.

EXAMPLE

```
CLS
EJECT 1
```

REMARK Also see SHUTDOWN.

MACINTOSH APPENDIX

FILES$ function

FORMAT

FILES$ *(Ø, "prompt" [, "default filename"] [, volume%])*
FILES$ *(1, "type list" [, volume%])*
FILES$ *(-9999,"","",volume%)*
FILES$ *(-expression,"", "Root Volume",volume%)*

DEFINITION

Returns the filename and volume number of a file selected by the user for saving (Ø) or loading (1). The beginning number (or expression) designates what information is being requested:

Ø — Returns information about a file to be saved. You may designate the "default filename" by using that option above. The volume number of where the file is to be saved is also returned (use it!).

1 — Returns information about the file to be loaded. You may allow the user to select only certain types of file by using "type list". For example; if you only wanted to load "TEXT" type files you would designate "TEXT". The volume number of the file is also returned and must be used in the OPEN statement if other than the current directory is chosen.

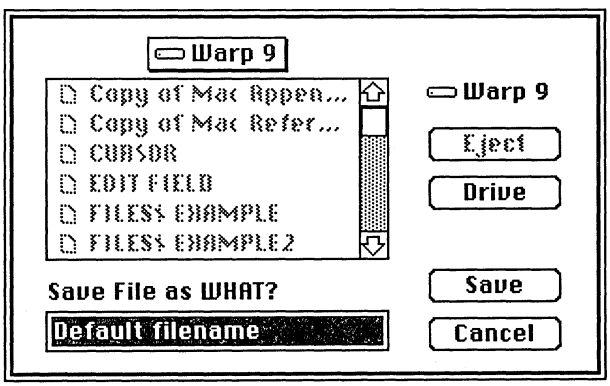
-9999 — Returns whether the current root volume is HFS or MFS. HFS is the most current type of system. MFS was the system used on older Macintoshes.

-expression — Returns the files and folders located in the "Root volume" designated. Volume number is also returned.

Examples:

SAVE A FILE

```
A$=FILES$ (Ø,"Save File as WHAT?", "Default filename",volume%)
```



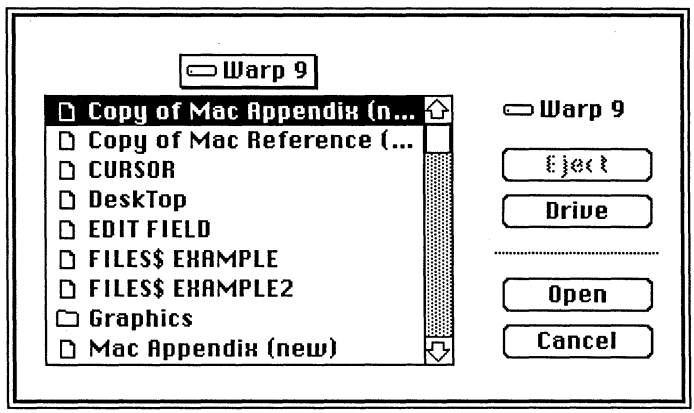
continued...

MACINTOSH APPENDIX

FILES$ continued....

LOAD A FILE

```
A$=FILES$ (1,"type", volume%)
```



EXAMPLE1

```
REM Example of LOAD and SAVE using FILES$
:
WINDOW#1, "FILES$"
DO
  LOCATE 0,0:CLS LINE:INPUT"<S>ave or <L>oad example of FILES$?";A$
  A$=UCASE$(A$)
  IF A$="S" THEN GOSUB"SAVE A FILE EXAMPLE"
  IF A$="L" THEN GOSUB "LOAD A FILE EXAMPLE"
UNTIL (A$<>"S") AND (A$<>"L"):
STOP
:
"SAVE A FILE EXAMPLE"
Filename$=FILES$(0,"Save File as WHAT?","Default filename",volume%)
SELECT Filename$
  CASE ""
    CLS:LOCATE 0,5:PRINT "Null string so SAVE cancelled!":BEEP
    DELAY 2000
  CASE ELSE
    CLS:LOCATE 0,5:PRINT"Filename to SAVE:";Filename$
    PRINT"The Volume number to save to:";volume%
    PRINT"File would be saved using:"
    PRINT TAB(30)" OPEN'A, O or R',1,Filename$,,volume%"
END SELECT
RETURN
:
"LOAD A FILE EXAMPLE"
Filename$=FILES$(1,"",,volume%)
```

continued...

MACINTOSH APPENDIX

FILES$ continued....

```
SELECT Filename$
  CASE ""
    CLS:LOCATE 0,5:PRINT "Null string so LOAD cancelled!":BEEP
    DELAY 2000
  CASE ELSE
    CLS:LOCATE 0,5:PRINT"Filename to LOAD is:";Filename$
    PRINT"The Volume number where file is:";volume%
    PRINT"File would be loaded using OPEN'I or R',1,Filename$,,volume%"
END SELECT
RETURN
```

EXAMPLE2

```
REM Example of getting pathnames and System types
REM Returns all the files and a complete PATH LIST
REM in INDEX$
CLEAR 10000 : L=0 : L2=0 : REM STORAGE FOR VOLUME$
FOR I=1 TO 1000
  A$=FILES$(-I,"",,V%):REM Get the Root volume
  LONG IF V%=0 OR A$=""
    I=1000:REM Quit looking for root volumes
  XELSE
    PRINT
    XFS$=FILES$(-9999,"","",V%)
    PRINTSTRING$(30,"*");" FILE SYSTEM ";XFS$;" ";STRING$(30,"*")
    PRINT "PATH NAME:";STRING$(56-POS(0),".");"FILE NAME"
    INDEX$(L)=A$ : L=L+1 : L2=L
    DO
      IF L > L2 THEN L2=L2+1 : A$=VOL$(L2-1)
      FOR J=1 TO 1000
        TRONX
        B$=FILES$(-J,"",A$,V%):REM Get folders/files in Root A$
        LONG IF B$=""
          J=1000
        XELSE
          PRINT A$;STRING$(56-POS(0),".");B$
          X$=RIGHT$(B$,1)
          LONG IF X$=":"
            INDEX$(L)=A$+B$ : L=L+1
          ENDIF
        ENDIF
      NEXT
    UNTIL L2 >= L
  ENDIF
NEXT : PRINT
PRINT STRING$(33,"*");" PATH LIST ";STRING$(33,"*")
FOR I=0 TO L-1
  PRINT I,INDEX$(I):REM Contains all Pathnames(not filenames)
  REM These pathnames could be used to search for filenames
  REM with their complete pathname.
NEXT
```

REMARK Inside Macintosh recommends that volume number be used whenever possible. This may be the only way to find a complete pathname.

MACINTOSH APPENDIX

FINDERINFO function

FORMAT **FINDERINFO** (*count%, var$* [(*n*)], *type&* [(*n*)], *volume%* [(*n*)])

DEFINITION Returns system information from the Finder to your application program so the program can do whatever is necessary with that file. This is useful when a user double clicks a file created with your application. The finder will load your application and pass it the information in FINDERINFO so you can determine if should be printed or loaded.

Example:

```
A%=FINDERINFO (count%, var$(n),  type&(n), volume%(n))
```

If A%=Ø the file is to be "Loaded". If A%=non-zero the file is to be printed.

count% Returns the number of files to be passed to your application. Sometimes people will click several files to be printed. The example program shows how to read them in. This also sets the maximum number of files accepted.

var$(n) A string variable that returns the filename(s) of the files being passed. A string array would be used if you wished to handle all the files being passed (each element MUST be defined as 31 chararacter length).

type&(n) The long word CVI of the file type(s). If a text file was being passed it would equal the CVI(TEXT). The type is created with DEF OPEN or CREATOR.

volume% (n) The disk volume numbers(s) where the filename(s) is located. If you wished to allow more than one file to be passed this would be an array.

EXAMPLE

```
DIM 31NAME$(1), VOL%(1), TYPE&(1)
DEFSTR LONG
COUNT% = 2
A% =FINDERINFO(COUNT%,NAME$(0),TYPE&(0),VOL%(0))
LONG IF COUNT% AND A%
  FOR C = 1 TO COUNT%
  OPEN I,#1,NAME$(C),,VOL%(0)
    DO
      LINE INPUT #1,A$
      PRINT A$
    UNTIL EOF(1)
  NEXT C
ENDIF
```

REMARK **Note:** String variable(s) must be defined as 31 in length. Also see DEF OPEN, BUNDLE and CREATOR.

MACINTOSH APPENDIX

FLUSHEVENTS statement

FORMAT **FLUSHEVENTS**

DEFINITION Deletes the contents of all the Event trapping FIFO buffers for DIALOG, BREAK, MENU MOUSE and TIMER.

EXAMPLE

```
WINDOW#1
FLUSHEVENTS
DIALOG ON:BREAK ON:MOUSE ON: MENU ON:TIMER ON
program continues...
```

REMARK It is recommended that you flush the events out of the queue BEFORE enabling event trapping.

MACINTOSH APPENDIX

GET FILE INFO function

FORMAT **GET FILE INFO** *variable$*

DEFINITION Returns system information about a file. See example.

EXAMPLE

```
WINDOW OFF: WINDOW #1,"GET FILE INFO example",,9
TEXT 4,9,0,1: DEFSTR LONG
:
REM  INITIALIZE POINTER TO 'FILE INFORMATION PARAMETER BLOCK'
hParmBlkPtr&=VARPTR(paramBlock$)
:
REM  TELL THE FILE MANAGER THERE IS NO 'IO' COMPLETION ROUTINE
ioCompletion&=0: POKE LONG hParmBlkPtr&+12,ioCompletion&
:
INPUT"Filename: ";filename$:ioNamePtr&=VARPTR(filename$)
POKE LONG hParmBlkPtr&+18,ioNamePtr&
:
REM  TELL THE FILE MANAGER TO USE THE 'DEFAULT' DRIVE
ioVRefNum%=0: POKE WORD hParmBlkPtr&+22,ioVRefNum%
:
REM  INPUT Volume number
INPUT"Input Volume number = ";InputioFDirIndex%
ioFDirIndex%=InputioFDirIndex%
POKE WORD hParmBlkPtr&+28,ioFDirIndex%
:
PRINT "VOLUME NUMBER = (<RETURN>=default): ";InputioFDirIndex%
:
GET FILE INFO paramBlock$
:
REM  GET (& COMPUTE) SELECTED VALUES FROM THE PARAMETER BLOCK
ioResult%=PEEK WORD(hParmBlkPtr&+16)
PRINT "Result code                          = ";ioResult%
ioNamePtr&=PEEK LONG(hParmBlkPtr&+18)
PRINT "Pathname                             = ";ioNamePtr&
PRINT "fileName                             =  ";filename$
ioVRefNum%=PEEK WORD(hParmBlkPtr&+22)
PRINT "Volume reference number              = ";ioVRefNum%
ioFRefNum%=PEEK WORD(hParmBlkPtr&+24)
PRINT "Path reference number                = ";ioFRefNum%
ioFDirIndex%=PEEK WORD(hParmBlkPtr&+28)
PRINT "Directory index                      = ";ioFDirIndex&
ioDirID&=PEEK LONG(hParmBlkPtr&+48)
PRINT "Directory ID or file number          = ";ioDirID&
ioFlStBlk%=PEEK LONG(hParmBlkPtr&+52)
PRINT "First allocation block of data fork  = ";ioFlStBlk%
ioFlLgLen&=PEEK WORD(hParmBlkPtr&+54)
PRINT "Logical end-of-file of data fork     = ";ioFlLgLen&
ioFlPyLen&=PEEK LONG(hParmBlkPtr&+58)
PRINT "Physical end-of-file of data fork    = ";ioFlPyLen&
ioFlRStBlk%=PEEK WORD(hParmBlkPtr&+62)
PRINT "First allocation block resource fork = ";ioFlRStBlk%
```

continued...

GET FILE INFO continued...

```
ioFlRLgLen&=PEEK WORD(hParmBlkPtr&+64)
PRINT "Logical end-of-file of Resource fork  = ";ioFlRLgLen&
ioFlRPyLen&=PEEK WORD(hParmBlkPtr&+68)
PRINT "Physical end-of-file of resource fork = ";ioFlRPyLen&
ioFlCrDat&=PEEKLONG(hParmBlkPtr&+72)
PRINT "Date and time of creation {internal#} =  ";UNS$(ioFlCrDat&)
longDate%=1: wantSeconds%=0
CALL IUDATESTRING(ioFlCrDat&,longDate%,date$)
CALL IUTIMESTRING(ioFlCrDat&,wantSeconds%,time$)
dateTime$=date$+" "+time$
PRINT "Date and time of creation             =  ";dateTime$
ioFlMdDat&=PEEKLONG(hParmBlkPtr&+76)
PRINT "Date/time last modified {internal#}   =  ";UNS$(ioFlMdDat&)
CALL IUDATESTRING(ioFlMdDat&,longDate%,date$)
CALL IUTIMESTRING(ioFlMdDat&,wantSeconds%,time$)
dateTime$=date$+" "+time$
PRINT "Date and time of last modification    =  ";dateTime$
PRINT:PRINT"Press a key...";
DO:UNTIL LEN(INKEY$)
END
```

REMARK Also see PUT FILE INFO in this appendix. See "Inside Mac" for specific technical information.

MACINTOSH APPENDIX

GET VOLUME INFO function

FORMAT **GET VOLUME INFO** *volume number%*

DEFINITION Returns system information about the volume specified by volume number.

EXAMPLE

```
WINDOW OFF:WINDOW #1,"GET VOL INFO example"
TEXT 4,9,0,1
:
REM  INITIALIZE POINTER TO 'VOLUME INFORMATION PARAMETER BLOCK'
hParmBlkPtr&=VARPTR(paramBlock$)
:
REM  TELL THE FILE MANAGER THERE IS NO 'IO' COMPLETION ROUTINE
ioCompletion&=0: POKE LONG hParmBlkPtr&+12,ioCompletion&
:
REM  INITIALIZE POINTER TO 'VOLUME NAME'
ioNamePtr&=VARPTR(volName$): POKE LONG hParmBlkPtr&+18,ioNamePtr&
:
REM  INPUT VOLUME REFERENCE NUMBER
REM  NOTE: 'DEFAULT' DRIVE VOLUME NUMBER is 0)
INPUT"Volume Number = ";InputioVRefNum%
ioVRefNum%=InputioVRefNum%: POKE WORD hParmBlkPtr&+22,ioVRefNum%
REM
REM  USE YOUR INPUT VALUE OF 'ioVRefNum'
ioVolIndex%=0: POKE WORD hParmBlkPtr&+28,ioVolIndex%
PRINT
PRINT "Volume Number = ";InputioVRefNum%
:
GET VOLUME INFO paramBlock$
:
REM  GET (& COMPUTE) SELECTED VALUES FROM THE PARAMETER BLOCK
:
ioResult%=PEEK WORD(hParmBlkPtr&+16)
PRINT "Result code                                = ";ioResult%
ioNamePtr&=PEEK LONG(hParmBlkPtr&+18)
PRINT "Pathname Pointer                           = ";ioNamePtr&
PRINT "Volume Name                                = ";volName$
ioVRefNum%=PEEK WORD(hParmBlkPtr&+22)
PRINT "Volume  number                             = ";ioVRefNum%
ioVolIndex%=PEEK WORD(hParmBlkPtr&+28)
PRINT "Volume Index                               = ";ioVolIndex&
ioVCrDate&=PEEKLONG(hParmBlkPtr&+30)
PRINT "Date/time of initialization (internal#)  = ";UNS$(ioVCrDate&)
longDate%=1: wantSeconds%=0
CALL IUDATESTRING(ioVCrDate&,longDate%,date$)
CALL IUTIMESTRING(ioVCrDate&,wantSeconds%,time$)
dateTime$=date$+" "+time$
PRINT "Date and time of initialization            = ";dateTime$
ioVLsMod&=PEEKLONG(hParmBlkPtr&+34)
```

continued...

MACINTOSH APPENDIX

GET FILE INFO continued...

```
PRINT "Date/time of last mod (INTERNAL#)          = ";UNS$(ioVLsMod&)
CALL IUDATESTRING(ioVLsMod&,longDate%,date$)
CALL IUTIMESTRING(ioVLsMod&,wantSeconds%,time$)
dateTime$=date$+" "+time$
PRINT "Date and time of last modification         = ";dateTime$
ioVNmFls%=PEEK WORD(hParmBlkPtr&+40)
PRINT "Number of files in directory               = ";ioVNmFls%
ioVNmAlBlks%=PEEK WORD(hParmBlkPtr&+46)
PRINT "Number of allocation blocks                = ";ioVNmAlBlks%
ioVAlBlkSiz&=PEEK LONG(hParmBlkPtr&+48)
PRINT "Size of allocation blocks                  = ";ioVAlBlkSiz&
ioVClpSiz&=PEEK LONG(hParmBlkPtr&+52)
PRINT "Default clump size                         = ";ioVClpSiz&
ioVNxtCNID&=PEEK LONG(hParmBlkPtr&+58)
PRINT "Next unused node ID                        = ";ioVNxtCNID&
ioVFrBlk%=PEEK WORD(hParmBlkPtr&+62)
PRINT "Number of unused allocation blocks         = ";ioVFrBlk%
freeBytes&=ioVAlBlkSiz&*ioVFrBlk%
PRINT "No. of free bytes                          = ";freeBytes&
ioVSigWord%=PEEK WORD(hParmBlkPtr&+64)
PRINT "Volume signature                           = ";ioVSigWord%
SigWord$=HEX$(ioVSigWord%)
volType$="error!"
IF SigWord$="D2D7" THEN volType$="MFS"
IF SigWord$="4244" THEN volType$="HFS"
PRINT "Volume Type                                =  ";volType$
ioVDrvInfo%=PEEK WORD(hParmBlkPtr&+66)
PRINT "Drive number                               = ";ioVDrvInfo%
ioVFSID%=PEEK WORD(hParmBlkPtr&+70)
PRINT "File-system identifier                     = ";ioVFSID%
ioVBkUp&=PEEKLONG(hParmBlkPtr&+72)
PRINT "Date and time of last backup (INTERNAL#)   = ";ioVBkUp&
CALL IUDATESTRING(ioVBkUp&,longDate%,date$)
CALL IUTIMESTRING(ioVBkUp&,wantSeconds%,time$)
dateTime$=date$+" "+time$
PRINT "Date and time of last backup               = ";dateTime$
ioVFilCnt&=PEEK LONG(hParmBlkPtr&+82)
PRINT "Number of files on volume                  = ";ioVFilCnt&
ioVDirCnt&=PEEK LONG(hParmBlkPtr&+86)
PRINT "Number of directories on volume            = ";ioVDirCnt&
PRINT:PRINT"Press a key to finish..."
DO:UNTIL LEN(INKEY$)
RETURN
```

REMARK See "Inside Mac" for more specific information.

MACINTOSH APPENDIX

GET WINDOW function

FORMAT **GET WINDOW** [*#*] *expr, var&*

DEFINITION Returns the LongInteger "Pointer" to the window specified by *expr* for use with TOOLBOX functions and calls.

EXAMPLE

```
WINDOW#1
CLS
GET WINDOW #1, WPtr&
PRINT "The handle of Window one=";WPtr&
END
```

REMARK See section call "TOOLBOX" in this appendix and "Inside Macintosh" for details.

Another alternative to this function is the WINDOW(14) function.

Also see WINDOW functions and statements and GrafPort in the QuickDraw section of the toolbox chapter of this appendix.

MACINTOSH APPENDIX

HANDSHAKE statement

FORMAT **HANDSHAKE** *port, type*

DEFINITION Sets up the handshaking type for the serial I/O with the OPEN"C" statement.

PORT

-1	Modem port
-2	Printer port

TYPE

-1	XON/XOFF Handshaking
0	No handshaking
1	CTS Hardware handshaking (Clear To Send)

EXAMPLE

```
OPEN"C",-2,9600
HANDSHAKE -2,-1          <---Set up for XON/XOFF handshaking
PRINT#-2, "Goodbye!"     <---Print directly to imagewriter (bypass Mac printer driver)
CLOSE#-2
END
```

REMARK See "Inside Mac" and the hardware manual for your serial device for specific technical information.

MACINTOSH APPENDIX

INDEX$ enhancements

FORMAT

INDEX$ (*expr* [,***Index#***])
INDEX$I (*expr* [,***Index#***])
INDEX$D (*expr* [,***Index#***])

INDEXF (*simplestring* [, *element#*] [, ***Index#***])

DEFINITION On the Macintosh INDEX$ may have up to 10 extra INDEX$'s from 0 to 9 (specified by index# above). Each INDEX array may be manipulated separately using the statements and functions above.

expr — expr may be a LongInteger value with the Mac version allowing INDEX$ arrays up to 2 billion elements (memory permitting).

Index# — Specifies a sub index of the INDEX Array i.e. INDEX$(0,1) and INDEX$(0,2) are two different INDEX$ arrays. You may specify the index# with all the regular INDEX$ statements and functions. Except for this they are identical to the regular INDEX$ (see INDEX$ in main reference).

Other Statements Affected by enhancments to INDEX$:

CLEAR n& [,*index#*]	If index# is used, this sets aside n& bytes of the INDEX$ array specified by index#. If index# is not used this statement is identical to the regular CLEAR statement as described in the main reference section.
CLEAR -1	Clears all INDEX$ arrays to zero (sets to null).
CLEAR INDEX$ [(*index#*)]	Clears only the INDEX$ array specified by index#.
m&=MEM [(*index#*)]	Returns memory available for that array.
m&=MEM (*index#*+10)	Returns number of active elements.
m&=MEM (*index#*+20)	Returns the length in bytes of the INDEX$ specified by index#.
m&=MEM (*index#*+30)	Returns the maximum number of bytes available in INDEX$.
m&=MEM (*index#*+40)	Returns the starting address of the INDEX$ specified (VARPTR)

continued....

MACINTOSH APPENDIX

INDEX$ enhancements continued...

EXAMPLE

```
CLEAR 10000
:
DEFTAB 7
:
INDEX$(1)="XXX"  : GOSUB "SHOW INDEX"
INDEX$(3)="TEST" : GOSUB "SHOW INDEX"
INDEX$D(2)       : GOSUB "SHOW INDEX"
CLEAR 1000       : GOSUB "SHOW INDEX"
:
STOP
"SHOW INDEX" : X&=MEM(40)
PRINT STRING$(35,"*");MEM(0);STRING$(35,"*")
PRINT "ADDR";X&,
PRINT "SIZE";PEEK LONG(X&-12),
PRINT "BYTES USED";PEEK LONG(X&-8),
PRINT "ELLEMENTS",PEEK LONG(X&-4)
:
FOR I=0 TO 10
 PRINT PEEK(X&+I),
NEXT : PRINT
RETURN
```

REMARK The enhancements to INDEX$ create opportunities for some powerful string handling and editing.

Be sure to see INDEX$ in the main reference section and in the "Variables" section of this manual for more information.

MACINTOSH APPENDIX

INKEY$ alternatives

FORMAT **DIALOG(16)** OR **CALL GETKEY(A%(n))**

DEFINITION This function operates the same as other versions of ZBasic accept during event trapping like BREAK ON, DIALOG ON, MENU ON, TIMER ON or MOUSE ON. During these events INKEY$ and the event types "Fight" over keys resulting in one or the other functions missing characters.

Since there are rare occassions when a Macintosh program does not use at least one of these events we felt it was imperative to provide a reliable way of intercepting keystrokes during events, thus; DIALOG(16) came to be.

When used during event trapping DIALOG(16) will return the ASCII character of the key:

EXAMPLE1

```
WINDOW#1,"Dialog(16) example of INKEY$"
DIALOG ON:BREAK ON
ON DIALOG GOSUB "Check Dialog"
:
"Main Event Loop"
GOTO "Main Event Loop"
:
DIALOG OFF:BREAK OFF
:
"Check Dialog"
D=DIALOG(0):D1=DIALOG(D)
IF D<>16 THEN RETURN:REM We're just checking for keys this time
PRINT CHR$(D1);
RETURN
```

EXAMPLE2

```
REM   This Example program displays the Macintosh keyboard status
REM   as keys are pressed.  This routine allows you to monitor
REM   the COMMAND, OPTION, CAPS LOCK, SHIFT and others!
:
DIM A%(7),B%(7)
CLS : FOR I=0 TO 7 : B%(I)=-1 : NEXT
"KEYSCAN"
CALL GETKEYS(A%(0))
LOCATE 0,0
FOR I=0 TO 7
  LONG IF B%(I)<> A%(I)
    LOCATE 0,I : CLS LINE : PRINT I,BIN$(A%(I)) : B%(I)=A%(I)
  ENDIF
NEXT
TRONX:REM Check for <Command Period>.
GOTO "KEYSCAN"
```

REMARK Also see INKEY$ in the main reference section of this manual.

Important Note: Obviously DIALOG(16) will be disabled when an EDIT FIELD is active since EDIT FIELD gets the keys.

MACINTOSH APPENDIX

KILL statement

FORMAT KILL *"filename" [, volume%]*

DEFINITION Same as the standard ZBasic KILL except that you may also specify the volume number where the file is located. The volume number may be found with FILES$.

EXAMPLE

```
WINDOW#1, "KILL"
file$=FILES$(1,"",,volume%)
CLS
PRINT"Are you sure you want to erase:"file$"??????"
INPUT"<Y>es or <N>o";A$
A$=UCASE$(A$)
IF A$<>"Y" THEN STOP
CLS
PRINT"OK, I am erasing:";file$
:
KILL file$,volume%
END
```

REMARK See KILL in the main reference section.

MACINTOSH APPENDIX

KILL PICTURE statement

FORMAT **KILL PICTURE,** *variable&*

DEFINITION Purges memory used by the PICTURE designated by variable&.

If many pictures are used (without using KILL PICTURE) you may get an out-of-memory system error.

EXAMPLE

```
WINDOW#1,"KILL PICTURE"
PICTURE ON:CALL SHOWPEN
CIRCLE FILL 512,383,180
FOR X=1 TO 3000
  PLOT RND(1024), RND(767) TO RND(1024),RND(767)
NEXT
PICTURE OFF, A&
Z&= MEM(-1)
KILL PICTURE A&
:
CLS
PRINT"BEFORE";Z&
PRINT"AFTER ";MEM(-1)
PRINT:PRINT"A total memory savings of:";MEM(-1)-Z&;" BYTES"
PRINT:PRINT:PRINT"Press a key to continue"
DO
UNTIL LEN(INKEY$)
END
```

REMARK Be careful to kill only pictures that exist and which will no longer be used.

Also see the MEMORY MONITOR desk accessory for viewing the memory used by PICTURE and other things.

MACINTOSH APPENDIX

LCOPY function

FORMAT **LCOPY**

DEFINITION Prints the contents of the current window to the Imagewriter.

EXAMPLE

```
WINDOW#1
DEF FNP&=PEEKLONG(PRHANDLE)
CIRCLE FILL 512,383,300
IF PEEK (FNP&+24)=1 THEN LCOPY
REM 1=Imagewriter, 3=Laserwriter
END
```

REMARK Also see PAGE LPRINT. May not function with Laserwriter printer drivers and in some cases may cause system errors.

See PRHANDLE and example above for determining the printer type installed on a system.

Also see ROUTE 128 for a more controllable way of printing window contents to a printer such as the LaserWriter.

MACINTOSH APPENDIX

LOF function

FORMAT **LOF** *(file# [, record length]*

DEFINITION Returns the number of records in a file. Same as the standard ZBasic LOF statement with an optional parameter to define the record length.

The second parameter is primarily useful for determining byte size of a file. LOF(1,1) would return the length of a file in bytes.

EXAMPLE

```
WINDOW#1
DO
  A$=FILES$(1,"",,volume%)
    OPEN"R",1,A$,,volume%
    PRINT"The file named '";A$;"' is ";LOF(1,1);" Bytes long"
    CLOSE#1
    PRINT:PRINT:PRINT"Press a key to continue"
    DO: UNTIL LEN(INKEY$)
UNTIL LEN(A$)=0
END

RUN

The file named 'Fred' is 23 Bytes long
```

REMARK Also see FILES$ and the LOF function in the main reference manual.

MACINTOSH APPENDIX

LPRINT statement

FORMAT **LPRINT** { % | @ } (*horizontal*, *vertical*)

DEFINITION Same as the standard ZBasic statement with the added enhancement of % and @ which work the same way as PRINT% and PRINT@ except on the ImageWriter™ or LaserWriter™.

LPRINT%(x,y) Prints at the relative coordinate position on the paper defined by x,y.

LPRINT@(X,Y) Prints at the character position on the paper defined by x,y.

EXAMPLE

```
WINDOW#1:WIDTH LPRINT=-2
CLS
DEF PAGE
DEF LPRINT
IF PRCANCEL THEN END
COORDINATE 850, 1100:REM Try others!
Y=20:X=150
DO:X=X+G1:G1=G1+1
  Y=Y+G:G=G+2
  LPRINT%(X,Y);"LPRINT%";
UNTIL X>700
:
Y=0
FOR X= 1 TO 63
  Y=Y+1
  LPRINT@(X,Y)"LPRINT@";
NEXT
CLEAR LPRINT
END
```

REMARK Also see ROUTE 128, DEF LPRINT, DEF PAGE, PRCANCEL, COORDINATE and the section in this appendix about printing with the "Imagewriter and Laserwriter".

Important Note: See WIDTH LPRINT-2 for a way of disabling text wrap-around checking and increasing print speed 4 to 10 times. Also see LPRINT in the main reference section.

MACINTOSH APPENDIX

MEM function

FORMAT

MEM ({ *-1* | *-2* })
MEM (*n*)

DEFINITION

MEM(-1) returns the maximum amount of available memory for the program and variables. Will force purging of all purgable memory blocks to determine total memory available.

May take from one to several seconds to complete depending on the condition of the memory.

If X&=MEM(-1); X& will return the amount of memory available. Be sure to use a LongInteger variable (regular integers may return a false value).

MEM(-2) returns the maximum number of contiguous free bytes on the current Heap.

MEM(n): See INDEX$ in this appendix.

EXAMPLE

```
WINDOW#1,"KILL PICTURE"
PICTURE ON:CALL SHOWPEN
CIRCLE FILL 512,383,180
FOR X=1 TO 3000
  PLOT RND(1024), RND(767) TO RND(1024),RND(767)
NEXT
PICTURE OFF, A&
Z&= MEM(-1)
KILL PICTURE A&
:
CLS
PRINT"BEFORE";Z&
PRINT"AFTER ";MEM(-1)
PRINT:PRINT"A total memory savings of:";MEM(-1)-Z&;" BYTES"
PRINT:PRINT:PRINT"Press a key to continue"
DO
UNTIL LEN(INKEY$)
END
```

REMARK

Be sure to read the sections on "Memory Management" and INDEX$ in this appendix.

May be a good idea to do this function at the start of your program to "Clean-up" memory.

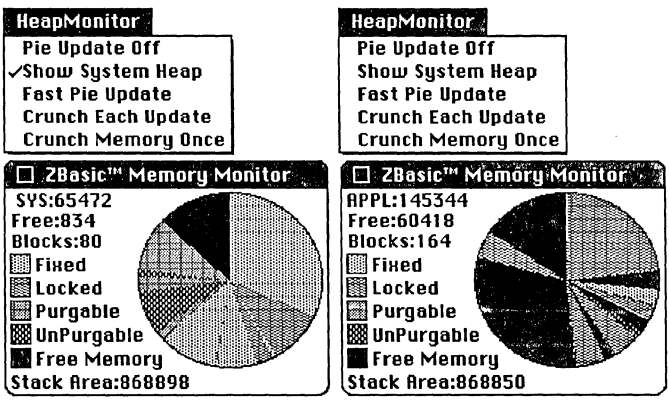
MACINTOSH APPENDIX

MEMORY MONITOR Desk Accessory

FORMAT

Memory Monitor

DEFINITION The Memory monitor DA has been to monitor system and application memory so you can see if there are problems with memory while your program is running! Use Font/DA Mover to install Memory Monitor on your diskettes (we've made it public domain).



REMARK

HEAP MONITOR MENU OPTIONS

Pie Update Off: Turns off the graphic pie chart updating (allows programs to run faster).
Show System Heap: Changes the display to System Heap vs. Application Heap
Fast Pie Update: Sets The Memory monitor to update the Memory Pie chart about once every 3 seconds. When off it updates about once every 10 seconds.
Crunch Each Update: Causes a MaxMem function to be called every update. All purgable blocks will be purged and relocatable blocks will be moved down in memory. Check if problems are occurring with handles or relocatable blocks.
Crunch Memory Once: Crunches memory like above but only once (same as doing MEM(-1)).

APPL: Application Heap Block Size. All parts of the current program(s) and variables must reside in this area. It may be expanded up to the Stack size.
SYS: System Heap Block Size. Operating System Heap.
Free: Bytes available for larger or new blocks. The Heap may expand up to Stack size less about 30,000 to 60,000 bytes.
Blocks: Blocks Presently in Use. Good way to find blocks that are allocated.
Stack Area: Free Stack memory If this number decreases rapidly---check things like: GOSUB without RETURN.

BLOCK TYPES

Fixed: Cannot be moved or purged until program is finished.
Locked: Same as fixed but may be UnLocked by the program.
Purgable: This type of block may be purged if its memory is required.
UnPurgable: This block may be moved but cannot be purged
Free Memory: This memory is available for more or larger blocks.

MACINTOSH APPENDIX

MENU function

FORMAT **MENU** *(expression)*

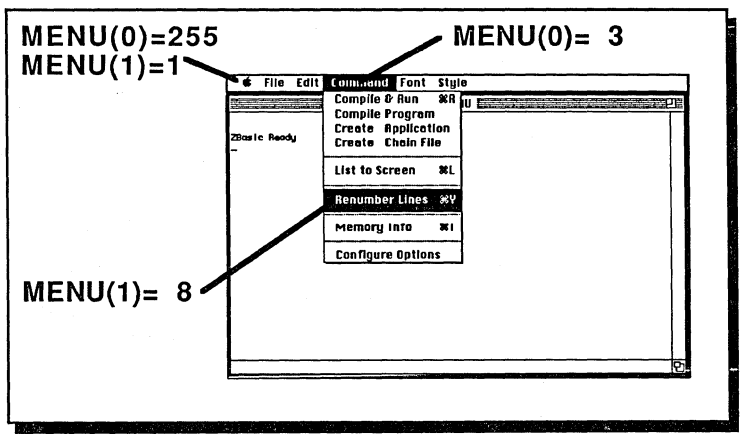
DEFINITION Using the two menu functions, MENU(Ø) and MENU(1), you can determine which menu item was selected.

These functions are returned when menu events take place between MENU ON and MENU OFF statements.

Your program can then call the appropriate routine using ON MENU GOSUB.

MENU(Ø) Returns the MENU number chosen on the menu bar and MUST be used BEFORE you use MENU(1). The menu returns 255 (used with APPLE MENU).

MENU(1) Returns the MENU item down which was selected. The menu returns 1 (used with APPLE MENU).



EXAMPLE See program example next page...

REMARK Note that both functions must be used as a pair. Selecting one without the other causes unpredictable results.

continued...

MACINTOSH APPENDIX

MENU function continued...

EXAMPLE

```
WINDOW #1, "MENU EXAMPLES"
COORDINATE WINDOW
:
APPLE MENU "<BAbout 'MENU EXAMPLES'"
EDIT FIELD#1, "Use with EDIT menu", (100,100)-(350,111),1,2
:
MENU 1, 0, 1, "Menu one"
MENU 1, 1, 64, "/1ITEM1 has @ sign because of ASCII 64"
MENU 1, 2, 2, "/2This item has a check mark!"
MENU 1, 3, 0, "(<B/3DISABLED MENU ITEM (GREY)"
MENU 1, 4, 1, "CLEAR THE SCREEN"
MENU 1, 5, 1, "/QQuit this program"
:
EDIT MENU 2
MENU 2,1,0,"Can't UNDO"
MENU 2,7,1,"Use this EDIT MENU with EDIT FIELDS!"
:
MENU 3, 0, 1, "Menu three"
MENU 3, 1, 1, "ITEM 1 OF 3... Notice how far the menu may go!"
MENU 3, 2, 1, "^1 Example of adding an ICON"
MENU 3, 3, 1, "<UITEM 3 OF MENU 3 is underlined (Meta char <U)"
MENU 3, 5, 1, "<BTHE 'BLACK BOX!'"
:
ON MENU GOSUB "Menu Check"
MENU ON: BREAK ON
"Main Event Loop"
GOTO "Main Event Loop"
MENU OFF:BREAK OFF
:
"Menu Check"
M1=MENU(0):M2=MENU(1)
PRINT@(0,0);: CLS LINE
PRINT "Item ";M2" of MENU";M1;" was chosen."
LONG IF M1=255 AND M2=1
  WINDOW#2,,(150,100)-(275,200),-3
  PRINT@(0,3)"This Example Program"
  PRINT@(0,4)"  by  Mike Gariepy"
  PRINT@(0,5)"   April 29, 1987"
  DO
  UNTIL MOUSE(3):REM Click mouse to close window
  WINDOW#1:WINDOW CLOSE#2
END IF
IF M1=1 AND M2 =4 THEN CLS
IF M1=1 AND M2 =5 THEN END
LONG IF M1=3 AND M2 =5
  BOX FILL 0,0 TO 1023, 767
  COLOR=0:BOX FILL 150,200 TO 320,230:COLOR=-1
  PRINT%(152,220)"The Black Box strikes Again!";
END IF
MENU:REM  Used to set the Menu back to normal
RETURN
END
```

MACINTOSH APPENDIX

MENU statement

FORMAT

MENU ON
MENU OFF
MENU STOP

DEFINITION Sets event trapping for lines between MENU ON and MENU OFF. A call will be made to the routine specified by ON MENU GOSUB.

MENU ON	Enables event trapping for MENU activity. This will dramatically slow program performance and you should use MENU OFF whenever high performance is required. Checks for MENU events at the beginning of each line.
MENU OFF	Disables event trapping for MENU activity. Use whenever a routine needs to execute at high speeds.
MENU STOP	Temporarily disables the ON MENU GOSUB statement and stores MENU events in a queue (buffer). Does not increase program performance. Use MENU OFF when high speed is required. Do another MENU ON to restore the ON MENU GOSUB. At that time the events in the queue will be processed. Note: This statement is automatically executed when a program jumps to a routine specified by ON MENU GOSUB and MENU ON is enabled again after a RETURN from the ON MENU GOSUB.

EXAMPLE See program example under MENU function.

REMARK See MENU function, APPLE MENU, EDIT MENU, ON MENU GOSUB and the program example under MENU function and the other MENU statements on the next page.

Important Note: Order of program lines determines where event trapping takes place... NOT the order of execution (this is different from the BASIC interpreter).

continued...

MACINTOSH APPENDIX

MENU statement (continued...)

FORMAT

MENU
MENU *menu number, item number, state* [, *title$*]

DEFINITION

MENU by itself resets the menu bar back to normal after a MENU event has taken place.

These statements are used to define the contents of pull-down menus. The optional parameters set characteristics of menu items:

menu number	The menu column (1-20) (across the top)
item number	The item number of the menu. If zero is specified, the Menu title is assumed (the items down from the top).
state	0=disables item 1=enable item 2=enable item with a check mark 3 or greater= puts that ASCII character in place of a check mark.
title$	The string name of the item. Use Meta-Characters to format text or add graphic icons.

USING META CHARACTERS TO FORMAT MENU CONTENTS

ZBasic allows meta characters to be used in *title$* to pass item formatting information to the MENU statement (note that the title on the menu bar cannot be changed , only the pull down items);

Meta-Character	Usage	
; or RETURN	Separates multiple items	
^	Followed with icon number, adds that icon to the menu	
!	Followed by a character, marks the menu item with that character. Such as a check mark.	
/	Followed by a character, associates the keyboard key equivalent that may be used with the control key.	
(	Disables the item (greys out). Will not allow user to select it.	
<	Followed by one of the following characters, sets the text attributes:	**<B** Bold faced **<I** Italicized **<U** Underlined **<O** Outlined **<S** Shadowed

EXAMPLE

See program example under MENU function

REMARK

Note: Any items skipped are replaced by disabled lines.

See information under MENU function, APPLE MENU, EDIT MENU, and the MENU statement on the previous pages. Also see the "Toolbox" section of this appendix.

MACINTOSH APPENDIX

MODE statement

FORMAT MODE [=]n

DEFINITION MODE is disabled (and ignored) with this version because it is not applicable. All Macintosh modes use graphics. All modes use high resolution graphics.

Character attributes may be changed using the TEXT statement.

All ZBasic print and graphic action takes place in WINDOWS, not on the screen like other versions of ZBasic. The upper-Left corner of a window is the 0,0 graphic and text coordinate position.

Graphic coordinates may be changed by using the COORDINATE statement. The default coordinates are 1023x767 as with all other versions of ZBasic. These coordinates are relative to the size of the window so if the window size is adjusted the coordinates will be adjusted at the same scale.

To get pixel coordinates use COORDINATE WINDOW.

EXAMPLE

```
MODE=7: REM Ignored....does absolutely nothing!
```

REMARK **CONVERTING ZBASIC PROGRAMS FROM OTHER COMPUTERS**

To make a Macintosh window emulate other computers you will need to set the text type to a mono-spaced font, like MONACO. The following attributes will set a typical window to approximate a 80 x25 character screen. Adjust the point size as necessary to your preferences:

TEXT 4, 9, 0, 0

Note: MODE uses the Apple ROMS to set the way the text replaces the background text and graphics. There is a bug in the older Macintosh ROMS (64K) that will print extra spaces after the text is printed on the screen. You may need to program around this problem when using the 64K ROM.

MACINTOSH APPENDIX

MOUSE function

FORMAT **MOUSE** (*expression*)

Important Note: This is an optional format to the standard ZBasic MOUSE functions (see main reference section). In order to enable this type of MOUSE function you need to use the DEF MOUSE=1 statement at the beginning of a program.

DEFINITION The following MOUSE(*expression*) functions return information about the MOUSE. The value of *expression* may be an integer number from zero to six.

MOUSE BUTTON EVENTS

MOUSE(Ø) This function returns a number of values depending on the MOUSE button event that has taken place

- **Ø** Button not pressed and has not been pressed since the last MOUSE(Ø) check.
- **1** Single-Click since last MOUSE(Ø) check. Use MOUSE(3), (4), (5) and (6) to check the first and last positions of the click.
- **2** Double-click since last MOUSE(Ø) check. Use MOUSE(3), (4), (5) and (6) to check the first and last positions of the double-click.
- **3** Triple-click since last MOUSE(Ø) check. Use MOUSE(3), (4), (5) and (6) to check the first and last positions of the double-click.
- **-1** Same as 1 but the mouse button is still being held down
- **-2** Same as 2 but the mouse button is still being held down.
- **-3** Same as 3 but the mouse button is still being held down.

Note: -1, -2, or -3 usually indicate a Click-and-drag event.

DETERMINING CURRENT MOUSE POSITION

MOUSE(1) Returns the last horizontal (X) coordinate of the mouse when the button was pressed.

MOUSE(2) Returns the last vertical (Y) coordinate of the mouse when the button was pressed.

---

continued...

MACINTOSH APPENDIX

MOUSE function continued...

DETERMINING MOUSE POSITIONS DURING CLICK EVENTS

MOUSE(3)	Returns the horizontal (X) coordinate of the mouse when the button was pressed the time BEFORE the last event. Very handy for determining where the mouse start was in a click-and-drag event.
MOUSE(4)	Returns the vertical (Y) coordinate of the mouse when the button was pressed the time BEFORE the last event. Very handy for determining where the mouse start was in a click and drag.
MOUSE(5)	Returns values depending on the status of MOUSE(Ø). If a button was pressed when MOUSE(Ø) was checked then this function returns the horizontal (X) position of the mouse. If the button was not being pressed then this function returns the horizontal position of the mouse when the button was released. May be used to calculate the last position of a "Click and drag".
MOUSE(6)	Returns values depending on the status of MOUSE(Ø). If a button was pressed when MOUSE(Ø) was checked then this function returns the vertical (Y) position of the mouse. If the button was not being pressed then this function returns the vertical position of the mouse when the button was released. May be used to calculate the last position of a "Click and drag".

REMARK See MOUSE statement in this appendix and the standard ZBasic MOUSE function in the reference section for optional use of the mouse.

Also see mouse tracking example in the toolbox chapter of this appendix.

Important note: Set DEFMOUSE=1 at the beginning of your program to enable this syntax. Otherwise the ZBasic MOUSE functions will be in affect.

Important note: The MSBASIC type MOUSE (DEF MOUSE=1) will not function outside of a window. Therefore be sure to open a window before attempting to use the MOUSE.

MACINTOSH APPENDIX

MOUSE statement

FORMAT

MOUSE ON
MOUSE OFF
MOUSE STOP

Important Note: This is an optional format to the standard ZBasic MOUSE functions (see main reference section). In order to enable this type of MOUSE statement you need to use the DEF MOUSE=-1 statement at the beginning of a program.

DEFINITION These statements enable or disable event trapping of MOUSE events

MOUSE ON	Begins checking for MOUSE events and returns those events in MOUSE functions (0-6). If an ON MOUSE GOSUB statement has been used a GOSUB will be made to the routine specified by line or label. When any event trapping is done there will be a significant reduction in program speed.
MOUSE OFF	Disables mouse event trapping. Be sure to turn off all event trapping when maximum program performance is required.
MOUSE STOP	Temporarily disables ON MOUSE GOSUB. Stores events in a queue and will return events again when MOUSE ON is used.

EXAMPLE

```
WINDOW#1,"MOUSE EXAMPLE"
DEF MOUSE=-1: REM Other MOUSE types
TEXT ,,,0: REM  Set for overwrite mode
MOUSE ON: BREAK ON
ON MOUSE GOSUB"Mouse Events"
:
"Main Event Loop"
GOTO "Main Event Loop"
MOUSE OFF
:
"Mouse Events"
XXX=MOUSE(0):X=MOUSE(1):Y=MOUSE(2)
DO:X2=MOUSE(1):Y2=MOUSE(2)
  PEN,,,10:REM Set XOR mode for graphics.
  PLOT X,Y TO X2,Y2
  PLOT X,Y TO X2,Y2
UNTIL MOUSE(0)=0
IF ABS(XXX)>1 THEN  PLOT X,Y TO X2,Y2
RETURN
```

REMARK

Important Note: MOUSE will not function in the ZBasic default window. You will have to open your own window before the event can be detected.

Important Note: Order of program lines determines where event trapping takes place... NOT the order of execution (this is different from the BASIC interpreter).

MACINTOSH APPENDIX

ON BREAK GOSUB statement

FORMAT **ON BREAK GOSUB** *LIne number or "label"*

DEFINITION When this statement is used and a BREAK event occurs, the program will call the routine specified by *line number or label.* A BREAK event occurs when the <COMMAND PERIOD> keys are pressed.

This statement is enabled only between BREAK ON and BREAK OFF program lines.

To disable this statement and still store BREAK events in the queue use BREAK STOP. After BREAK STOP you can re-enable the event trapping by executing another BREAK ON at which time any events in the queue will be handled.

EXAMPLE

```
WINDOW#1
DIALOG ON:BREAK ON
ON BREAK GOSUB "BREAK OUT"
ON DIALOG GOSUB "Dialog Event"
:
"Main Event Loop"
GOTO "Main Event Loop"
DIALOG OFF
:
"Dialog Event"
PRINT"DIALOG(0)=";DIALOG(0)
PRINT"DIALOG(1)=";DIALOG(1)
RETURN
:
"BREAK OUT"
END
```

REMARK See BREAK statements and the section "Writing Macintosh Programs" in this appendix.

Important Note: For breaking out of programs the ON BREAK GOSUB is not necessary. A Simple BREAK ON near the beginning of a program will cause an automatic scan for the <COMMAND PERIOD> keys. If these keys are pressed an END statement will be executed.

Also see TRON, TRONB, TRONX, TRONV, TRONS and TRON WINDOW for other ways of breaking out of programs (for best results don't mix BREAK and TRON).

MACINTOSH APPENDIX

ON DIALOG GOSUB statement

FORMAT **ON DIALOG GOSUB** *LIne number or "label"*

DEFINITION When this statement is used and a DIALOG event occurs, the program will call the routine specified by *line number or label.* See DIALOG function for the types of events that are trapped with this statement.

This statement is enabled only between lines containing DIALOG ON and DIALOG OFF.

To disable this statement and still store events in the queue use DIALOG STOP. After a DIALOG STOP you can re-enable the event trapping by executing another DIALOG ON at which time any events in the queue will be handled.

EXAMPLE

```
WINDOW#1
DIALOG ON:BREAK ON
ON DIALOG GOSUB "Dialog Event"
:
"Main Event Loop"
GOTO "Main Event Loop"
DIALOG OFF
:
"Dialog Event"
PRINT"DIALOG(0)=";DIALOG(0)
PRINT"DIALOG(1)=";DIALOG(1)
RETURN
```

REMARK See DIALOG ON, DIALOG OFF, DIALOG STOP statements, the DIALOG function and the section "Writing Macintosh Programs" in this appendix.

Important Note: An automatic DIALOG STOP is executed when this statement is used. Upon RETURN an automatic DIALOG ON is executed. This prevents the routine from calling itself and causing system errors.

MACINTOSH APPENDIX

ON MENU GOSUB statement

FORMAT **ON MENU GOSUB** *LIne number or "label"*

DEFINITION When this statement is used and a MENU event occurs, the program will call the routine specified by *line number or label.*

This statement is enabled only between lines containing MENU ON and MENU OFF.

To disable this statement and still store MENU events in the queue use MENU STOP. After a MENU STOP you can re-enable the event trapping by executing another MENU ON at which time any events in the queue will be handled.

EXAMPLE

```
WINDOW#1
MENU ON: BREAK ON
ON MENU GOSUB "Menu Event"
:
MENU 1, 1, 1, "/QQuit"
:
"Main Event Loop"
GOTO "Main Event Loop"
MENU OFF
:
"Menu Event"
X=MENU(0):Y=MENU(1)
IF X=1 AND Y=1 THEN END
RETURN
```

REMARK See MENU ON, MENU OFF, MENU STOP statements, the MENU function and the section "Writing Macintosh Programs" in this appendix.

Important Note: An automatic MENU STOP is executed when this statement is used. Upon RETURN an automatic MENU ON is executed. This prevents the routine from calling itself and causing system errors.

MACINTOSH APPENDIX

ON MOUSE GOSUB statement

FORMAT **ON MOUSE GOSUB** *LIne number or "label"*

DEFINITION When this statement is used and a MOUSE event occurs, the program will call the routine specified by *line number or label.*

This statement is enabled only between lines containing MOUSE ON and MOUSE OFF.

To disable this statement and still store MOUSE events in the queue use MOUSE STOP. After a MOUSE STOP you can re-enable the event trapping by executing another MOUSE ON at which time any events in the queue will be handled.

WARNING: Use DEF MOUSE=1 or BOMB!.

EXAMPLE

```
WINDOW#1:DEF MOUSE=1
MENU ON: MOUSE ON:BREAK ON
ON MOUSE GOSUB "Mouse Event"
ON MENU GOSUB "Menu Event"
:
MENU 1, 1, 1, "/QQuit"
:
"Main Event Loop"
GOTO "Main Event Loop"
MENU OFF:MOUSE OFF
:
"Menu Event"
X=MENU(0):Y=MENU(1)
IF X=1 AND Y=1 THEN END
RETURN
:
"Mouse Event"
FOR X=0 TO 6
  PRINT"MOUSE EVENT";X;"=";MOUSE(X)
NEXT
RETURN
```

REMARK See MOUSE ON, MOUSE OFF, MOUSE STOP statements, the MOUSE function and the section "Writing Macintosh Programs" in this appendix.

Important Note: An automatic MOUSE STOP is executed when this statement is used. Upon RETURN an automatic MOUSE ON is executed. This prevents the routine from calling itself and causing system errors.

MACINTOSH APPENDIX

ON TIMER (n) GOSUB statement

FORMAT **ON TIMER (n) GOSUB** *LIne number or "label"*

DEFINITION This statement is used to set up a call to a specific routine every ***n*** seconds.

This statement is enabled only between lines containing TIMER ON and TIMER OFF.

If n=10 then the program will be interrupted every ten seconds.
IF n=600, the program will be be interrupted every 10 minutes.

EXAMPLE

```
WINDOW#1
MENU ON: TIMER ON:BREAK ON
ON TIMER 10 GOSUB "Timer Event"
ON MENU GOSUB "Menu Event"
:
MENU 1, 1, 1, "/QQuit"
:
"Main Event Loop"
GOTO "Main Event Loop"
MENU OFF:TIMER OFF
:
"Menu Event"
X=MENU(0):Y=MENU(1)
IF X=1 AND Y=1 THEN END
RETURN
TIMER OFF
:
"Timer Event"
J&=TIMER
PRINT "LAST TIMER";I&
PRINT "THIS TIMER";J&
PRINT "--------------------------"
PRINT "TIME ELAPSED";J&-I&
I&=J&
RETURN
```

REMARK Also see TIMER statement and function in this appendix.

Note: This statement need not be enabled to get the current value of TIMER.

MACINTOSH APPENDIX

OPEN statement

FORMAT OPEN "[{ **R** | **D** }] {**A** | R | O | I }", *filenumber, "filename" [, record length][,**Volume%]***

DEFINITION The OPEN statement for the Macintosh is the same as the standard ZBasic OPEN statement with the following enhancements:

Types	Definition
R	RESOURCE FORK: Used when you want to open the Resource fork of a file. The Resource fork of an application, like ZBasic, contains the object code and resources. The DATA FORK of that application could be used to store FONT or WINDOW SIZE preferences for that application (or whatever).
D	DATA FORK: This is the default. Plain DATA files usually have empty Resource forks. You could use the Resource fork for storing other important important information about that file (like two files in one).
A	Used to OPEN a file for APPEND. This opens the file and sets the file pointer to the end of the file so subsequent PRINT# or WRITE# statements will add data at the end. OPEN "RA" for resource fork and "DA" for data fork.
Volume%	Tells the system which file volume number is to be used to save the file. The volume number is returned from the FILES$ function. This way of designating the volume is much better than using pathnames and is recommended in "Inside Macintosh".
filename	Pathnames may also be used in the filename with colon separators. Volume number is preferable in most cases.

EXAMPLE

```
OPEN"DO",1,"FRED.TXT"
PRINT#1,"Hello"
CLOSE#1
:
OPEN"DA",1,"FRED.TXT"
PRINT#1,"Goodbye"
CLOSE#1
:
OPEN "I",1,"FRED.TXT"
DO
  INPUT#1,A$
  PRINTA$
UNTIL EOF(1)
END

RUN

Hello
Goodbye
```

REMARK Also see GET FILE INFO, PUT FILE INFO, GET VOLUME INFO and FILES$ in this appendix. Also see OPEN, ON ERROR GOSUB and the chapter "Disk Errors" in the main reference section.

MACINTOSH APPENDIX

OPEN "C" statement

FORMAT **OPEN "C",** *-1 or -2 [,[baud][,[parity][,[stopbit][,word length], [buffer]]]]*

DEFINITION Enhancements to the standard ZBasic statement include:

-1	Modem port
-2	Printer port
baud	Up to 19,200, and possibly 57,600 baud for some applications. *buffer* may need to set larger for higher*baud* rates. Note that 38,400 baud is also supported for MIDI fans.
stopbit	2=1.5 stop bits
word length	2=5 bits 3=6 bits
buffer	This version adds the "*buffer*" enhancement. Default buffer length is 64 bytes. Minimum buffer Length is 1 byte. Maximum length is 32,767 bytes.

EXAMPLE

```
CLS
REM Example of printing text directly to the Imagewriter.
REM This by-passes the Macintosh printer driver and allows
REM the user to pass control codes that would be stripped out
REM by that driver if you used LPRINT or ROUTE 128.
:
OPEN"C",-2,9600
HANDSHAKE=-1:REM XON/XOFF handshaking
PRINT#-1,CHR$(27);CHR$(31);"Hello"
CLOSE#-2
END
```

REMARK Also see HANDSHAKE in this appendix .

See main reference section for more information about OPEN "C" and some example terminal programs and file transfer programs.

Important Note: Don't mix OPEN"C" with LPRINT when printing. OPEN"C" won't print to the ImageWriter or LaserWriter through AppleTalk. Note that some versions of BASIC use LPRINT to route output directly to the printer while ZBasic sends LPRINT output to the current printer driver (selected from "Chooser" DA).

MACINTOSH APPENDIX

OPEN TALK statement

FORMAT **OPEN TALK** , *- ref# , " [string] "[, buffer size]*

DEFINITION This statement opens the MacinTalk speech synthesis software driver and allows you to specify Phoneme or American speech commands and set the size of the buffer.

The Macintalk driver takes ASCII text and converts it into the synthetic speech sounds of a male, non-regional, standard American. There is also optional phoneme support.

The MacinTalk™ program from Apple is included with the master ZBasic diskette and must be in the SYSTEM folder of the application using itor a system error may occur (simply copy the system icon called "Macintalk" to the system folder from the ZBasic master disk).

-ref# A negative number describes the using MacinTalk with a PRINT # -ref, *string* . Be sure to use a number different than others used as reference numbers for Serial ports, AppleTalk Network ports or OPEN HELP.

"string" Normally use "". This is reserved for an "exception file". If "noReader" is used, you may use phonemes instead of English (default is English).

buffer size Creates a buffer for storing speech text. The longer the text the longer the buffer size needs to be. Default is 256 bytes.

CLOSE # -ref Closes the Macintalk file.

EXAMPLE

```
REM Example of using ASCII text (regular English words)
OPEN TALK,-1"",512
TALK#-1, 150,110,0
:
DO
  INPUT"Tell me what to say: ";Speak$
  PRINT#-1, Speak$
UNTIL LEN(Speak$)=0
CLOSE#-1
:
REM Example of using Phonemes (special symbols representing human
REM voice sounds.  This lets you create more realistic speech).
OPEN TALK,-1"noReader",512
TALK#-1, 150,110,0
PRINT#-1, "/HEH3LOW"      :REM Remember: Phonemes only!  No ASCII.
CLOSE#-1
```

REMARK Important: Always CLOSE MacinTalk when finished.

See TALK statement and the section "MacinTalk" in the front of this appendix for more information.

MACINTOSH APPENDIX

PAGE LPRINT function

FORMAT **PAGE LPRINT**

DEFINITION Prints the contents of the current window to the imagewriter.

EXAMPLE

```
WINDOW#1
:
REM     Note that a function is used to get the PRHANDLE each time
REM     This is because it moves around in memory and may not be
REM     in the same place for long.
:
DEF FNP&=PEEKLONG(PRHANDLE)
CIRCLE FILL 512,383,300
IF PEEK (FNP&+24)=1 THEN PAGE LPRINT
REM 1=Imagewriter, 2=Daisywriter, 3=Laserwriter
END
```

REMARK Also see LCOPY. Does not function with the Laserwriter and in some cases may cause system errors. See PRHANDLE and example above for determining the printer type installed on a system.

See ROUTE 128 for alternate ways of sending text and graphics to a LaserWriter.

Same as pressing SHIFT-OPTION-4. PAGE LPRINT is the same syntax as the MS-DOS™ version of ZBasic and functions the same as LCOPY.

MACINTOSH APPENDIX

PEN statement

FORMAT **PEN** *[Xsize][, [Ysize], [visible], [pen mode], [pen pattern]]*

DEFINITION PEN is used to specifiy the height and width of the pen point, the mode of how the pen interacts with the background, the color and pattern of lines, circles and boxes created with ZBasic commands and quickdraw routines (like rounded rectangles and polygons):

Xsize, Ysize The height and width of the pen point (respectively). See example. Note: The upper left corner of the "PEN" is the coordinate pointer.

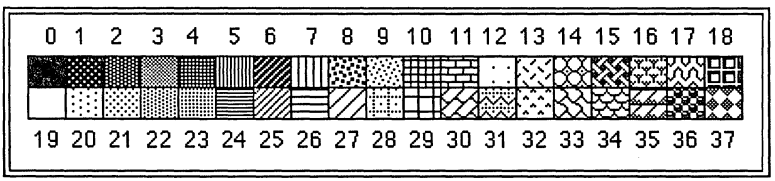
visible Visible= non-zero and Invisible= zero.

pen mode Sets the transfer mode through which the pen pattern is transfered onto the screen when shapes are drawn. The modes are:

8	PATTERN COPY	9	PATTERN OR
10	PATTERN XOR	11	PATTERN BIC
12	NOT PATTERN COPY	13	NOT PATTERN OR
14	NOT PATTERN XOR	15	NOT PATTERN BIC

MODES	DEFINITION
COPY	Replaces the pixels in the destination area.
OR	Replace black with black, white with white.
XOR	XORs pixels. Great for animation.
BIX	BIC erases black pixels to white.

pen pattern Patterns (below). Patterns may be changed from the control panel.



EXAMPLE

```
WINDOW#1,"PEN Example": COORDINATE WINDOW
:
CLS:PRINT"Press a key to see the X thickness of PEN"
DO:UNTIL LEN(INKEY$)
FOR X1=3 TO 475 STEP 22
  X=X+1:Y=60:PRINT%(X1-3,45);X;
  PEN X,1: PLOT X1,Y TO X1,310
NEXT
DO:UNTIL LEN(INKEY$)
CLS:PRINT"Press a key to see the Y thickness of PEN"
DO:UNTIL LEN(INKEY$):Y=0
FOR Y1=15 TO 375 STEP 16
  Y=Y+1:X=60:PRINT%(40,Y1+9);Y;
  PEN 1,Y:PLOT X,Y1 TO 450,Y1
NEXT
DO:UNTIL LEN(INKEY$)
```

REMARK See COORDINATE, PLOT, CIRCLE and BOX for ways of drawing using the PEN created formats. See the "Toolbox" section of this appendix for other ways of using PEN.

MACINTOSH APPENDIX

PICTURE function

FORMAT **PICTURE**

DEFINITION Returns a LongInteger handle (a memory address pointer) to the current picture area specified by PICTURE ON. If PICTURE is used by itself it means that the current picture is to be used. Pictures may be stored with LongInteger pointers using PICTURE OFF, *var&* or by *var&*=PICTURE. Pictures may be put to the window with: PICTURE, *var&*.

EXAMPLE1

```
REM SAVE a PICTURE,var& to a RESOURCE FILE
WINDOW#1, "PICTURE Example"
:
REM This routine creates a PICTURE called Picture&
PICTURE ON: CALL SHOWPEN
CIRCLE 512,384,350
PLOT 0,0 TO 1023,767:PLOT 1023,0 TO 0,767
PICTURE OFF
Picture&=PICTURE
:
PICTURE, Picture&
CALL CREATERESFILE ("Testfile")
Resourcenumber%=1:REM Store several pictures if you want.
A%=FN OPENRESFILE ("Testfile")
CALL ADDRESOURCE (Picture&, CVI("PICT"), Resourcenumber%, "")
CALL CLOSERESFILE (A%): KILL PICTURE, Picture&
```

Note: A number of pictures may be stored in the same file and may be specified by resource number% (an integer number from 1 to 32,767).

EXAMPLE2

```
REM LOAD a PICTURE,var& from a RESOURCE FILE
A%=FN OPENRESFILE("Testfile")
Resourcenumber%=1: REM You may have many pictures
Picture&=FN GETPICTURE (Resourcenumber%)
REM
REM  Use CALL DETACHRESOURCE (handle) only if you want to
REM  keep the picture in your program. Otherwise you must
REM  show the Picture BEFORE you CLOSE the resource file
REM  with CALL CLOSERESFILE.
REM
CALL DETACHRESOURCE (Picture&)
CALL CLOSERESFILE(A%)
PICTURE, Picture&
```

Note: Resourcenumber% may be used to store or load many pictures in one resource file. These numbers MUST be unique (do not duplicate numbers within a resource file). See the Macintosh toolbox call: UNIQUEID.

REMARK See Next Page for an example of creating and using a PICTURE bigger than the screen. Also see PICTURE statement, KILL PICTURE, WIDTH -2 and PEN. A PICTURE is limited to 32K.

continued...

MACINTOSH APPENDIX

PICTURE function continued...

EXAMPLE3

```
REM Example of creating a PICTURE bigger than the screen.
REM CALL CLIPRECT allows this. It uses the first 4 integer
REM variables to get the coordinates: T,L, B and R.
REM See CALL CLIPRECT in the toolbox section for more info.
:
DIM T,L,B,R : T=0 : L=0 : B=1024 : R=1024
WIDTH -2:REM Allow printing without WRAP around IN WINDOW.
DEF MOUSE=-1:REM Mac MOUSE
WINDOW OFF:APPLE MENU "Instructions to 'The BIG PICTURE'"
WINDOW#1,"BIG PICTURE",(10,40)-(510,340),1
COORDINATE WINDOW:       REM WINDOW area = 500 x 300
CALL CLIPRECT(T):        REM CLIP   area = 1024 x 1024
PICTURE ON:TEXT,,,0
PLOT 0,0 TO 1023,767: BOX 0,0 TO 1024,1024
CIRCLE FILL 512,512,300
PRINT%(10,1014)"Bottom Left"
PRINT%(900,1014)"Bottom Right"
PRINT%(10,20)"Upper left"
PRINT%(900,20)"Upper Right"
PRINT%(500,512)"CENTER"
PICTURE OFF:P&=PICTURE
:
MENU 1,0,1,"File"
MENU 1,1,1,"/QQuit"
MOUSE ON:MENU ON
ON MENU GOSUB"Menu":ON MOUSE GOSUB"Mouse"
X=0:Y=0:GOSUB "Show Picture"
:
"Main Event Loop"
GOTO "Main Event Loop"
MOUSE OFF:MENU OFF
:
"Show Picture"
PICTURE(X,Y),P&: RETURN
:
"Menu"
M=MENU(0):M1=MENU(1)
IF M=1 AND M1=1 THEN END
IF M<>255 THEN MENU:RETURN
WINDOW#2,"Instructions: 'The Big Picture'",(100,100)-(400,300),24
PRINT:PRINT:PRINT
PRINT"To examine different parts of the picture"
PRINT"press the MOUSE button and drag the window"
PRINT"to the position you want"
DELAY 5000: BEEP:
WINDOW#1:WINDOW CLOSE#2:MENU:RETURN
:
"Mouse"
XX=MOUSE(0):I=MOUSE(1):J=MOUSE(2)
DO
  X=X+I-MOUSE(1):Y=Y+J-MOUSE(2)
  IF X<-539 THEN X=-539 ELSE IF X>0 THEN X=0
  IF Y<-739 THEN Y=-739 ELSE IF Y>0 THEN Y=0
  CLS:GOSUB "Show Picture" :DELAY 25
UNTIL MOUSE(0)=0:PRINT"X=";X" Y="Y"
RETURN:END
```

MACINTOSH APPENDIX

PICTURE statement

FORMAT

PICTURE ON *[(x1,y1) - (x2,y2)]*
PICTURE OFF *[,var&]*
PICTURE *[(x1,y1)][-(x2, y2)][,var&]*

DEFINITION

PICTURE may be used to store program Graphics, QuickDraw and/or Text activity between a PICTURE ON and PICTURE OFF statement. These pictures can be recalled with the Longinteger& variable handle.

DIFFERENCE BETWEEN PICTURE and GET/PUT GRAPHICS
PICTURE provides a means of storing graphic images as symbols and coordinates instead of pixel images. This requires much less storage and provides a way of doing graphics in the finest quality of the current device (like the Laserwriter). GET and PUT graphics are fast but require much more storage. An example of the storage difference of say 30 circles on the screen with GET would be about 20K while PICTURE would only require about 200 bytes.

Picture activity will not be seen unless you use CALL SHOWPEN. CALL HIDEPEN may also be used to hide graphics activity (default).

EXAMPLE

PICTURE ON	Stores the graphic or text into the current PICTURE buffer. The entire window area is assumed. See previous page to create a PICTURE area larger than the screen.
PICTURE ON (0,0)-(40,40)	Only activity occurring in this "region" will be stored.
PICTURE OFF ,Pict&	Stops storing graphics and text in the PICTURE buffer. Will use Pict& as a handle to this picture.
PICTURE (x1,y1)- (x2,y2), var&	This statement draws a PICTURE at the coordinate specified by x1,y1. IThe picture is scaled (enlarged or reduced) to fit into the coordinates specified by x2,y2. To force the picture to be go off the edge of the window you will need to give x2,y2 coordinates outside the window coordinates.
PICTURE, OldPicture&	Draws OldPicture to present cursor position.
PICTURE (10,10)-(45,100)	Fits picture into coordinates specified
PICTURE (0,0),a&	Draws picture specified by a& at 0,0 and scales the picture to fit in the window.
PICTURE -(50,50), Pic1&	Draws picture at original position and scales it to fit into the x2,y2 parameters specified.

REMARK

Note: A single PICTURE is limited to 32K. See examples on previous pages. Also see PICTURE function and KILL PICTURE.

MACINTOSH APPENDIX

PRCANCEL function

FORMAT **PRCANCEL**

DEFINITION When PRCANCEL returns true (non zero) the user has pressed the CANCEL button in the DEF LPRINT or DEF PAGE dialog box. This means the user wants to cancel printing and you may want to abort a print routine.

EXAMPLE

```
WINDOW#1,"PRCANCEL Example"
:
DEF PAGE
IF PRCANCEL PRINT"CANCEL PRESSED!":BEEP
:
DEF LPRINT
LONG IF PRCANCEL=0
  GOSUB"PRINT ROUTINE"
XELSE
  PRINT"CANCEL PRESSED! Printing Aborted!"
  BEEP:BEEP:DELAY 2000:END
END IF
END
:
"PRINT ROUTINE"
LPRINT"HELLO"
CLEAR LPRINT
END
```

REMARK Also see DEF PAGE and DEF LPRINT.

Note that <COMMAND PERIOD> will force the print manager to abort printing the current page, even if BREAK ON is not enabled.

MACINTOSH APPENDIX

PRHANDLE function

FORMAT **PRHANDLE**

DEFINITION This special function returns a pointer to the Macintosh printer parameter tables. Using this pointer you can retrieve information about the page and print set-ups.

EXAMPLE

```
WINDOW#1,"PRHANDLE Example",(10,100)-(500,300)
DEFDBL INT P: DEF PAGE
REM Get PRHANDLE (It doesn't stay in the same place in memory)
DEF FN P=PEEKLONG(PRHANDLE)
LONG IF PRCANCEL
PRINT"CANCEL PRESSED!":BEEP
END IF
DEF LPRINT
LONG IF PRCANCEL
PRINT"CANCEL PRESSED! Printing Aborted!"
BEEP:BEEP:DELAY 2000:END
XELSE
GOSUB"Page Set-up Information"
END IF
GOSUB"Print Info": CLEAR LPRINT
END
:
"Page Set-up Information"
REM Page Rectangle Height/Width in points
PRY=PEEKWORD(FNP+8):PRX=PEEKWORD(FNP+10)
PRY2=PEEKWORD(FNP+12):PRX2=PEEKWORD(FNP+14)
REM Paper Dimensions Height/Width in points
PDY=PEEKWORD(FNP+16):PDX=PEEKWORD(FNP+18)
PDY2=PEEKWORD(FNP+20):PDX2=PEEKWORD(FNP+22)
REM Print Manager version
PM=PEEKWORD(FNP)
REM Driver Info
DI=PEEKWORD(FNP+2)
REM Vertical and Horiz Resolution
VR=PEEKWORD(FNP+4):HR=PEEKWORD(FNP+6)
REM Paper Heigth/Width (coordinates)
PH=PEEKWORD(FNP+26):PW=PEEKWORD(FNP+28)
REM Printer Port 1=MODEM, 2=PRINTER
PP=PEEKWORD(FNP+30)
REM Printer Type 1=Imagewriter, 2=Laser, 3=Laser+
PT=PEEK(FNP+24)
REM First Page, Last Page
FP=PEEKWORD(FNP+62):LP=PEEKWORD(FNP+64)
REM Number of Copies
LONG IF PT>1
NC=PEEKWORD(FNP+46)
XELSE
NC=PEEKWORD(FNP+66)
END IF: RETURN
```

continued...

MACINTOSH APPENDIX

PRHANDLE statement continued...

```
"Print Info":CLS
PRINT"PRHANDLE=",,FNP
PRINT"PAGE RECT",,PRX","PRY,PRX2","PRY2
PRINT"PAPER RECT",,PDX","PDY,PDX2","PDY2
PRINT"PRINT MANAGER",,PM
PRINT"DRIVER=",,DI
PRINT"Vert/Horiz res",,VR,HR
PRINT"Paper Height/Width",PH,PW
PRINT"PRINTER PORT",,PP
PRINT"PRINTER TYPE",,PT
PRINT"FIRST/LAST PAGE",,FP,LP
PRINT"Number of copies",NC
PRINT:PRINT"Any key to continue..."
DO
UNTIL LEN(INKEY$):RETURN
```

REMARK See *Inside Macintosh"* and the Toolbox section of this appendix for more information about the printer handle.

```
****************** PRHANDLE DATA STRUCTURE ********************

PEEK Size (PEEK LONG(PRHANDLE)+Offset)

Offset   Size    -- Name --  -------- Description -----------
0        word    iPrVersion  Printing software version Number
2        word    iDev        Used Internally
4        word    iVRes       Vert Resolution Dots/Inch
6        word    iHRes       Horz Resolution Dots/Inch
8        Rect    rPage       Page Rectangle (Printable Area)
16       Rect    rPaper      Paper rectangle, Actual Paper Size
24       word    wDev        Device hi-Byte: 1=Imagewrite, 3=Laser
26       word    iPageV      Vert Paper size
28       word    iPageH      Horz Paper size
30       byte    feed        Paper feeder type (Sheet Feed etc.)
32       Record  prInfoPT    Print Time Imaging metrics (Record?)
(46)     word    iCopies     Copies to Print (Laserwriter only)
46       word    iRowBytes   (with imagewriter only)
48       word    iBandV      See "Inside Macintosh"
50       word    iBandH      See "Inside Macintosh"
52       word    iDevBytes   Bytes Required For Spooling
62       word    iFstPage    First Page to Print
64       word    iLstPage    Last  Page to Print
66       word    iCopies     Copies to Print (imagewriter)
68       byte    bJDocLoop   Print quality: 0=Draft, 1=Spool
69       byte    fFromApp    Printing from an Application
70       Ptr     pIdleProc   Idle Procedure While Printing
74       ptr     pFileName   Spool File Name:    NIL for default
78       word    iFileVol    Spool File volume
80       byte    bFileVers   Spool File version

* See "Inside Macintosh" for specifics.
```

MACINTOSH APPENDIX

PRINT USING statement

FORMAT **USING** "...###.###...^^^^";*numeric expression*

DEFINITION In addition to the normal PRINT USING formats provided by standard ZBasic, this version provides a new parameter for expressing formatted scientific notation.

The new parameter that may be used is the carot (^).

EXAMPLE

```
INPUT "Enter a number";X#
PRINT USING"+#### ^^^^  ";X#
PRINT USING"+####.### ^^^^   ";X#
PRINT USING"+###.### ^^^^  ";X#
PRINT USING"+##.### ^^^^  ";X#
PRINT USING"+.## ^^^^  ";X#
PRINT USING"+.# ^^^^  ";X#

Enter a number;?123.456

+1234E-01
+1234.560E-01
+123.456E+00
+12.346E+01
+1.235E+02
+.123E+03
+.12E+03
+.1E+03
```

REMARK If you are using exponents greater than 99, leave one space after the last carot (^) for each digit of exponent exceeding 99.

Note that some BASIC languages use a "D" to show double precision exponent such as: .123D-01. ZBasic always uses an "E".

Note: Exponents may be up to five digits: E±16,383

See PRINT USING in the main reference section for other USING options.

MACINTOSH APPENDIX

PUT statement (graphics)

FORMAT PUT *(x1,y1)* **[-(x2,y2)]** *,var(n),* **mode**

DEFINITION The same as the standard ZBasic PUT statement with the following additions.

The -(x2,y2) parameter may be included to scale bit images. They may be expanded or shrunk to your preference and within the capabilities of the hardware.

Along with the standard XOR, OR, AND, PSET and PRESET modes, mode for the Macintosh may also be a number between 0 and 7:

HOW BACKGROUND AND FOREGROUND INTERACT

MODE	DEFINITION
0	COPY
1	OR
2	XOR
3	BIC
4	NOT COPY
5	NOT OR
6	NOT XOR
7	NOT BIC

EXAMPLE

```
DIM A(10000)
COORDINATE WINDOW
:
CIRCLE 100,100,80
GET (0,0)-(100,100),A(0)
:
CLS
DO
  PUT (0,0)-(100+X,100+Y),A(0)
  DELAY 100
  PUT (0,0)-(100+X,100+Y),A(0),2:REM XOR mode
  X=X+2:Y=Y+2
UNTIL X>400 OR Y>300
END
```

REMARK Also see GET and PUT in the main reference section of this manual. See "Inside Macintosh" for detailed descriptions of the COPY, BIC, and other graphics "modes".

Be sure to see GET and PUT in the main reference section for more information.

MACINTOSH APPENDIX

PUT FILE INFO statement

FORMAT **PUT FILE INFO** *paramBLOCK$*

DEFINITION Puts file information retried and modified by the user back to the file.

EXAMPLE See GET FILE INFO for example

REMARK See *PBHSetFInfo* in "Inside Macinosh" for detailed technical information.

MACINTOSH APPENDIX

READ FILE statement

FORMAT **READ FILE** [*#*] *filenumber, destinationAddress&, NumberofBytes&*

DEFINITION An extremely fast way of loading data from a disk file. Reads the number of bytes specified by *NumberofBytes&* to the address specified by *destinationAddress&* from the file specified by *filenumber.*

EXAMPLE1

```
REM Program will load and display a MACPAINT file
DIM X%(1),71 X$(719) : X%(0)=576 : X%(1)=720
:
A$=FILES$(1,"PNTG",,V%): IF A$="" THEN STOP
OPEN"I",1,A$,1,V% : FL&=LOF(1)-512 : RECORD #1,512
:
A&=VARPTR(A$) : Y&=VARPTR(X$(0)) : X&=A& : N=256
FOR I=1 TO 720 : REM Lines in a MACPAINT Picture
  LONG IF N>180
    BLOCKMOVE X&,A&,256-N : X&=A&
    IF N>FL& THEN NX=FL& ELSE NX=N
    READ FILE #1,A&+256-N,NX : FL&=FL&-NX
  END IF
  REM Puts bit image in memory.
  CALL UNPACKBITS(X&,Y&,72):N=X&-A&
NEXT
CLOSE#1
PUT (0,0),X%(0),PSET: REM PUT PAINT picture on screen.
END
```

EXAMPLE2

```
REM  This routine uses WRITE FILE to save
REM  a MacPaint image. Use with example above
:
DEF OPEN "PNTG": REM Set file type to: PNTG
:
F$=FILES$(0,"MacPaint Save as..",F$,V%)
IF F$="" THEN RETURN
W0=1 : OPEN"O",1,F$,1,V% : A$=STRING$(128,0)
CURSOR=4 : WRITE #1, A$;128, A$;128, A$;128, A$;128
X&=VARPTR(X$(0)) : A&=VARPTR(A$) : AS&=A&
FOR I=1 TO 720
  CALL PACKBITS(X&,A&,72) : N=A&-AS&
  LONG IF N>180 OR I=720
    WRITE FILE #1,AS&,N : A&=AS&
  ENDIF
NEXT I
CLOSE#1
RETURN: REM Use as a subroutine
```

REMARK See OPEN, CLOSE and the WRITE FILE# statement for way of saving data to file.

MACINTOSH APPENDIX

RENAME statement

FORMAT RENAME *filename1$ TO filename2$* [, ***volume number%***]

DEFINITION Same as the Standard ZBasic RENAME statement except that *volume number%* may be used as the last parameter to specify the folder or volume location of the file whose name will be changed.

EXAMPLE

```
Name$=FILE$(1,"",,VOLUME%)
RENAME Name$ TO "Fred", VOLUME%
END
```

REMARK See FILES$ in this appendix and RENAME in the main reference.

MACINTOSH APPENDIX

ROUTE 128 statement

FORMAT **ROUTE 128**

DEFINITION Same as the Standard ZBasic ROUTE 128 statement except that with the Macintosh both graphics and text are routed.

EXAMPLE1

```
WINDOW#1,"ROUTE 128 example"
:
DEF LPRINT:                          REM Set-up Printer
IF PRCANCEL THEN STOP:               REM STOP if CANCEL pressed.
:
COORDINATE 1020,1340 :               REM Set Paper coordinates
:
ROUTE 128:                           REM Output to the printer
TEXT 0,36,64
PRINT"HELLO NUMBER ONE"
CIRCLE FILL 510,670,400
TEXT 0,9,0
PRINT%(900,1000)"Hello number two!!"
PEN 10,10
BOX 0,0 TO 1020,1340:PEN 1,1
:
ROUTE 0:                             REM Output back to the screen
CLEAR LPRINT
END
```

EXAMPLE2

```
REM Example of ROUTING Graphics and text in COLOR to an
REM ImageWriter II with a COLOR ribbon
DEF LPRINT
ROUTE 128
FOR I=0 TO 7
  COLOR I
  PRINT"THIS IS IN COLOR";I : PRINT
  BOX FILL 800,I*18 TO 814,I*18+16
  CIRCLE 512,384,I*32
  CIRCLE 768,I*32+32,32
NEXT
STOP
```

REMARK Also see LPRINT, TEXT, PEN, DEF PAGE, DEF LPRINT, and PICTURE.

MACINTOSH APPENDIX

RUN statement

FORMAT **RUN** *filename*

DEFINITION Execute another application from ZBasic.

EXAMPLE

```
RUN "MacPaint"
```

REMARK For example of executing other programs while retaining variables see WRITE FILE example in this reference section.

Also see RUN command, function and statement in the main reference section and the section on "Executing ZBasic programs" in the front of this manual.

MACINTOSH APPENDIX

SCROLL statement

FORMAT **SCROLL** *(x1,y1)-(x2,y2), amount x, amount y*

DEFINITION To cause scrolling of text and graphics in the current output window use the **SCROLL** statement. The syntax is:

x1,y1	specifies the upper left corner
x2,y2	specifies the lower right corner
amount x	specifies the horizontal amount of movement in pixels. If a negative number is used the movement is to the left. If positive numbers are used the movement is to the right.
amount y	specifies the vertical amount of movement in pixels. If a negative number is used the movement is up. If positive numbers are used the movement is down.

EXAMPLE

```
CLS
X1=-12:Y1=-12
CIRCLE FILL 512, 383, 50
:
FOR X=1TO30
  X1=X1+1:Y1=Y1+1
  SCROLL (0,0)-(1023,767), X1,Y1
NEXT
```

REMARK Scrolling may done in any direction. To scroll up and to the right you would give a negative y and a positive x. Your program must update scrolled areas of the screen.

See example of SCROLL in the example program with SCROLL BUTTON.

Note: Care should be taken to update position when scrolling windows with buttons or scroll bars.

MACINTOSH APPENDIX

SCROLL BUTTON statement

FORMAT **SCROLL BUTTON**[#] ref, current [[, min] , max][,[page] [, (x1,y1)-(x2,y2)][, type]]]

DEFINITION ZBasic allows you to create the familiar Macintosh™ scroll bars for your programs. Note that you MUST use ref numbers different from those used with the BUTTON statement.

Lets you create the standard vertical scroll bar (type 1), the horizontal scroll bar on the bottom of the window (type 2), and position and size your own scroll bars (type Ø).

button number The number used to identify a specific Scroll bar. Numbers are integer and must be one or greater and MUST NOT have the same numbers as the BUTTON statement.

current If the min value is Ø and the max value is 1000 and you give a current value of 500, the scroll box will appear in the middle. With a value of Ø, the box would appear at the top (or left).

min Sets the minimum value for scroll bar positioning (-32768 min).

max Sets the maximum value for scroll bar positioning (+32767 max).

page If the scroll bar is clicked between the arrow and the box the current value (returned in BUTTON(button number)) is incremented (or decremented) by this much. Don't use negative numbers.

(x1,y1)-(x2,y2) Tells ZBasic where to position the movable scroll bar (type Ø). Note that the type Ø scroll bar width and height may be changed. This parameter is ignored when using type 1 and 2 scroll bars. The coordinates are ZBasic coordinates. If the height is greater than the width, a vertical scroll bar will be displayed. If the width is greater than the height a horizontal scroll bar will used.

type There are three types of SCROLL BARS:

- **Ø** Movable. May be sized with rectangle coordinates. Works with all Macintosh window types. You may also set the width and height of this type of scroll bar . Experiment...
- **1** Vertical window scroll bar. Positioned on left from the top of the window to the bottom of the window and is sized automatically if the window size is changed. X-Y coordinates are ignored. Window types 1 and 5 only.
- **2** Horizontal scroll bar. Positioned on the bottom from the left of the window to the right of the window and are sized automatically if the window size is changed. X-Y coordinates are ignored. Window types 1 and 5 only.

REMARK To set up event handling for Scroll bars use ON DIALOG . Note: Normal scroll bars are 16 pixels wide.

continued...

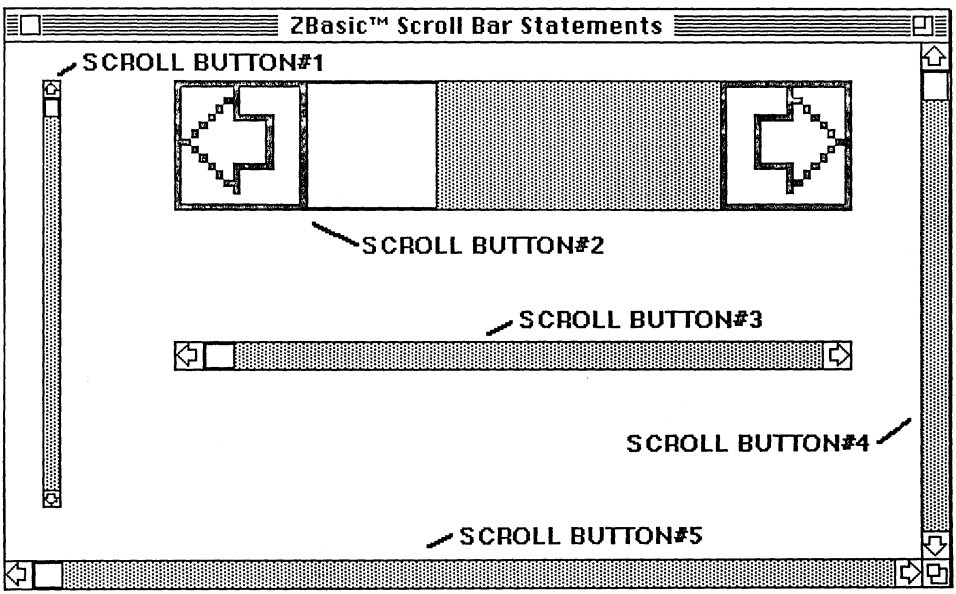
MACINTOSH APPENDIX

SCROLL BUTTON continued...

EXAMPLE1

```
WINDOW#1, "ZBasic™ Scroll Bar Statements"
COORDINATE WINDOW:WINDOW OFF: DIALOG ON: BREAK ON
DO:UNTIL DIALOG(0)=0:ON DIALOG GOSUB "DIALOG"
:
REM  Customizable Scroll bars (type zero):
SCROLL BUTTON #1, 0, 0, 32000, 1000, (20,20) -(30,250) ,0
SCROLL BUTTON #2, 0, 0, 32000, 1000, (50,20) -(420,90) ,0
SCROLL BUTTON #3, 0, 0, 32000, 1000, (50,160)-(420,176),0
:
REM  Regular standard Macintosh Vertical and
REM  Horizontal Scroll bars (types 1 and 2):
SCROLL BUTTON #4, 0, 0, 32000, 1000, ,1:REM Vertical
SCROLL BUTTON #5, 0, 0, 32000, 1000, ,2:REM Horiz
:
"Main Event Loop"
GOTO "Main Event Loop"
BREAK OFF:DIALOG OFF
"DIALOG": END
```

OUTPUT:



continued...

MACINTOSH APPENDIX

SCROLL BUTTON statement continued

EXAMPLE

```
REM   Text Window Scroll Bar(s) Example
REM   ZBasic 3.05 or Greater 1/22/87 A.G. ZEDCOR, INC.
WINDOW #1,"SCROLL BAR EXAMPLE", (20,60)-(500,340),1
COORDINATE WINDOW :WIDTH -2 :X=MEM(-1):REM Disable Line Wrap
WINDOW 1,"Example of Scrolling text files",(50,50)-(450,255),9
TEXT 4,9 :DIM A,D,W,L
CALL GETFONTINFO(A) :H=A+D+L :REM Get Font Size
"GOWAY"
A$=FILES$(1,"TEXT",,V%) : IF A$="" THEN BEEP : STOP ELSE CLS
OPEN "I",#1,A$,1,V% : CLEAR LOF(1)+32 : TL=0 : IF MEM=0 THEN STOP
WHILE NOT EOF(1) : REM Read TEXT file into INDEX$ array
  LINEINPUT#1,W$ : INDEX$(TL)=W$ : TL=TL+1 :REM Fill INDEX$
WEND
CLOSE : OV=0 : OH=-1 : SS=0 : REM Set SS =1 for Smooth-Scrolling
:
SCROLL BUTTON 1,OV,OV,TL-1,TL/10,,1
SCROLL BUTTON 2,OH,OH,255,10,,2
:
ON DIALOG GOSUB "DIALOG" : WINDOW#1,A$ : REM Filename/Titlebar
DIALOG ON : BREAK ON: REM Var 'SS' 0=Normal Scroll 1=Soft Scroll
:
"Main Event Loop"
GOTO "Main Event Loop"
:
DIALOG OFF: BREAK OFF
"DIALOG": D=DIALOG(0)
ON D GOTO
"BUTTON","X","ACTIVE","GOWAY","UPDATE","X","X","ZOOM","ZOOM"
"ACTIVE" : WINDOW #DIALOG(D) : RETURN : REM Activate this Window
"BUTTON" : IF DIALOG(D)=1 THEN BV=BUTTON(1)  ELSE "SIDE"
:
REM For "SOFT" scroll change ABS(X)>1 to ABS(X)>SL (next line)
X=OV-BV  : IF ABS(X) > 1 THEN OV=BV : CLS : GOTO "UPDATE"
:
IF X>0THEN DV=H:DL=-1:L=0:P=A ELSE DV=-H:DL=+1:L=SL+1:P=(SL-1)*H+A
WHILE OV<>BV : IF SS THEN DV=SGN(DV) : II=1 ELSE II=H
  FOR II=II TO H
    SCROLL (0,0)-(W6,W7),,DV : REM SCROLL 1 line or 1 Pixel
    PRINT %(-BUTTON(2)*W,P+(DV*(II-H)));INDEX$(OV-1+L);
  NEXT : OV=OV+DL
WEND   : RETURN
"ZOOM" : CLS : RETURN : REM ERASE IF ZOOM-IN OR ZOOM OUT
"SIDE" : SCROLL(0,0)-(W6,W7),(OH-BUTTON(2))*W,0:OH=BUTTON(2)
"UPDATE" : W6=WINDOW(6)-1 : W7=WINDOW(7)-1 : SL=W7/H
FOR II= OV TO OV+SL-1 : REM ******* Re-Draw Full Screen *******
  PRINT %(-BUTTON(2)*W,(II-OV)*H+A);INDEX$(II);
NEXT
COLOR 0: BOX FILL 0,SL*H TO W6,W7 :COLOR -1 :REM  Erase Bottom
"X"
RETURN
REM ************************************************************
REM To do smooth pixel scrolling change:
REM SS=1 and change ABS(X) > 1 to  ABS(X) > SL
REM Add 'CLEAR 0' to Erase INDEX$ Array When Window Closed.
REM ************************************************************
```

MACINTOSH APPENDIX

SEGMENT statement

FORMAT **SEGMENT**

DEFINITION Forces the compiler to end a memory segment. SEGMENT forces the current segment to end and start a new segment, thus the segments are arranged according to the physical location of the code, not the logical flow of the code.

The SEGMENT RETURN statement causes that segment of memory to be "purgable" and it will be discarded if memory is at a premium. This allows for large programs to be executed in computers with less memory.

Segments will automatically end if they reach about 28K bytes. The Macintosh memory manager allows segments up to a maximum of about 32K. ZBasic tries to break segments at "safe" places.

EXAMPLE

```
X=X+1: GOSUB "Other routine"
.
.
.
SEGMENT
"Other routine"
PRINT "Hello, this is the 'Other routine'"
X1=X1+22
SEGMENT RETURN
SEGMENT
.
.
.
```

Program lines above between "Other Routine" and SEGMENT RETURN are defined as a separate segment (between SEGMENT statements). SEGMENT RETURN is the same as a regular RETURN statement except that it "unloads" that segment, i.e. if memory is needed, the segment is discarded. If memory is not needed, the segment remains.

REMARK MEM (-1) purges "unloaded memory segments" and returns the memory remaining for segments. Use COMPILE to get the length of each segment. See MEM -1, COMPILE, ZBASIC MEMORY MONITOR Desk Accessory and SEGMENT RETURN.

CAUTION: The programmer must calculate the sizes of segments and "Manage" the use of segments in memory so that loading a segment into the system without sufficient memory does not occur. If you do load a segment into memory that is to large, a "System Error" will occur. See COMPILE for segment size.

CAUTION: If a segment break occurs in the middle of a FOR/NEXT, LONG/IF, SELECT/END SELECT or any other structure, the performance of a program will decrease dramatically. In some cases system errors may occur. If you are creating very large programs, you can monitor the position of default SEGMENT breaks while using the COMPILE or LCOMPILE commands.

Segments may be any size. Small segments will not alter the performance of a program unless there are lots of calls outside of that segment to other segments.

See "Memory Manager" section in the front of this appendix for more information.

MACINTOSH APPENDIX

SEGMENT RETURN statement

FORMAT **SEGMENT RETURN**

DEFINITION Same as the RETURN statement except that it tells the Macintosh "Memory Manager" that SEGMENT is no longer being used and that it is "Purgable"; thus if memory is at a premium (perhaps on an old Macintosh with 128K), that segment MAY be purged from memory freeing up room for other segments to be loaded and used.

If memory is not at a premium the segment will not be disturbed (on Macintoshes with plenty of memory, say ; a 512K Mac or a Mac II with 8 giga-bytes [gigglebytes?]).

EXAMPLE

```
X=X+1: GOSUB "Other routine"
.
.
.
SEGMENT
"Other routine"
PRINT "Hello, this is the 'Other routine'"
X1=X1+22
SEGMENT RETURN
SEGMENT
.
.
.
```

REMARK Remember that the program **must** return to another segment other than the one being unloaded; otherwise a "System Error" will occur.

See COMPILE to determine the size of segments.

MEM (-1) deletes purgable memory segments and returns the memory remaining for other segments. See SEGMENT, MEM(-1), ZBASIC MEMORY MONITOR Desk Accessory and COMPILE for more information. For detailed technical information about the Macintosh memory manager see "Inside Macintosh".

See "Memory Manager" section in the front of this appendix for more information.

MACINTOSH APPENDIX

SHUTDOWN statement

FORMAT **SHUTDOWN**

DEFINITION Ejects all diskettes and goes ***PING!***

Restarts the computer as if you just turned it on.

EXAMPLE

```
WINDOW #1,"SHUTDOWN Example"
CLS
FOR X=1 TO 1000
  PRINT"BYE!     ";
NEXT
SHUTDOWN
END
```

REMARK Also see EJECT.

MACINTOSH APPENDIX

SOUND function

FORMAT **SOUND**

DEFINITION Returns the number of sounds in the SOUND buffer.

SOUND=0 Sound buffer empty.
SOUND=16 Sound buffer full.

EXAMPLE

```
SOUND WAIT
FOR I=1 TO 4
  SOUND 400,100,1
  SOUND 800,200
  SOUND 400,100
  PRINTI,SOUND
NEXT
INPUT A$
PRINT"RESUME",SOUND
SOUND RESUME
SOUND WAIT
SOUND 1000,100
SOUND 100,500
INPUT A$
SOUND RESUME
DO
  TRONX
  S=SOUND
  PRINT S,
UNTIL S=0
```

REMARK Also see SOUND statement and WAVE.

MACINTOSH APPENDIX

SOUND statement

FORMAT

SOUND *frequency, duration* ***[,volume], [voice]***
SOUND WAIT
SOUND RESUME

DEFINITION

Same as standard ZBasic SOUND statement with the following enhancements:

volume	Sets the volume of the sound output. Range is 0 to 255. Note: Actual volume is controlled from the control panel DA.
voice	Values from 0 to 3 signifying which voice to use. The Macintosh has four voice capability. Default is zero.
SOUND WAIT	Causes sound output to wait until SOUND RESUME.
SOUND RESUME	Restarts sound output of current buffer.

EXAMPLE

```
DIM A%(255) : CLS
FOR I=0 TO 255
  A%(I)=RND(32)+I/4
NEXT
WAVE 3,A%(0) : WAVE 0
PRINT"RANDOM WAVE SOUNDS"
FOR I=100 TO 800 STEP I
  SOUND I,500
NEXT
PRINT"4 VOICE SOUNDS"
FOR I=1 TO 8
  WAVE 1 : SOUND WAIT
  FOR J=0 TO 3
    READ F : SOUND F,400,,J : PRINTF,
  NEXT
  SOUND RESUME : PRINT
NEXT
DATA 400, 400, 400, 000, 400, 400, 400, 000
DATA 400, 800, 800, 000, 500, 500, 100, 200
DATA 100, 200, 400, 800, 100, 110, 120, 130
DATA 400, 450, 500, 550, 000, 000, 000, 400
PEN ,,,10
DO
  FOR I=0 TO 255 STEP 4
    CIRCLE FILL 512,384+192,192 TO I,4
  NEXt: TRONX
UNTIL SOUND=0
```

REMARK

Also see SOUND function and WAVE in this appendix and SOUND in main reference section of this manual.

Note: The maximum volume is controlled with the Control Panel desk accessory.

MACINTOSH APPENDIX

SYSERROR function

FORMAT **SYSERROR**

DEFINITION Returns the number of the corresponding Macintosh system error:

SYSTEM ERRORS

General Errors

0	No error has occurred
-1	Queue element not found during deletion
-2	Invalid queue element
-3	Core routine number out of range
-4	Unimplemented core routine

I/O System Errors

-17	Control error
-18	Status error
-19	Read error
-20	Write error
-21	Bad Unit error
-22	Unit empty error
-23	Open error
-24	Close error
-25	Tried to remove an open driver
-26	Couldn't find driver in resources
-27	I/O call aborted by KILLIO
-28	A driver has not been opened

Memory Allocation Errors

-108	Not enough room in heap zone
-109	Handle was nil in handle zone
-111	WhichZone failed (applied to free block)
-112	Trying to purge a locked block
-113	Address in zone check failed
-114	Pointer check failed
-115	Blcok check failed
-116	Size check failed

System Errors

32767	Undefined error
1	Bus error
2	Address error
3	Illegal instruction
4	Divide by zero
5	Check exception
6	Trap-V Exception
7	Privilege violation
8	Trace exception
9	Line 1010 exception (Macbug trap)
10	Line 1111 exception (Macbug trap)
11	Misc. exception
12	Unimplemented routine
13	Spurious interrupt
14	I/O system error
15	Segment Loader error
16	Floating point error (SANE)
17	Can't Load Package #0
18	Can't Load Package #1
19	Can't Load Package #2
20	Can't Load Package #3
21	Can't Load Package #4
22	Can't Load Package #5
23	Can't Load Package #6
24	Can't Load Package #7
25	Can't Allocate Mem block
26	Segment loader error
27	File Map Destroyed
28	Stack Overflow Error
29	Unknown
30	"Please Insert disk"

Numbers not shown had not been defined by Apple as of 5/87 (there may be others).

REMARK See "Inside Macintosh" for detailed explanations of these error messages.

Also see "Runtime and Compiletime Errors", ERROR and ON ERROR GOSUB in the main reference section for more information.

MACINTOSH APPENDIX

TALK statement

FORMAT **TALK***[#] -ref# , speed, pitch, mode (natural or robotic), sex*

DEFINITION This statement is used to change the characteristics or the sounds of the MacinTalk voice synthesizer speech. Be sure that TALK has been opened and is on the same volume.

speed	Varies the rate of speech . Numbers may range from 85 to 425 words per minute. Default is 150 words per minute.
pitch	Pitch of the voice in the range of 65 to 500 Hz. Default is 110 Hz.
mode	Ø=Natural. 1=Robotic.
sex	Apple says; "Available for future implementation"

EXAMPLE

```
OPEN TALK,-1"",512
FOR speed=85 TO 425 STEP 20
  TALK#-1, speed,110,0
  PRINT#-1, "Hello"
NEXT
CLOSE#-1
:
OPEN TALK,-1"",512
FOR pitch=65 TO 500 STEP 20
  TALK#-1, 150,pitch,1
  PRINT#-1, "Hello"
NEXT
CLOSE#-1
```

REMARK See OPEN TALK statement and the section "MacinTalk" in this appendix for more information.

Note: Make sure the Macintalk driver is installed in the current SYSTEM folder. A file error will occur if not found.

MACINTOSH APPENDIX

TEHANDLE function

FORMAT TEHANDLE

DEFINITION Returns the handle to the current EDIT FIELD so the programmer can manipulate text in the fields larger than 255 characters.

EXAMPLE

```
REM This example by Andrew Gariepy shows you how to use
REM the INDEX$ string with TEHANDLE to manipulate EDIT
REM FIELD strings greater than 255 characters.
:
DIM LargeRect,x,y2,x2 : y2=8192 : x2=8192 : CLEAR 10000,0
COORDINATE WINDOW : MENU 1,0,1,"File" : EDIT MENU 2
WINDOW #1 : TEXT ,,,0
EDIT FIELD #1,"HELLO",(5,70)-(480,270),2,1
:
DEF FN teWord%(n,o)=PEEK WORD(PEEK LONG(TEHANDLE(n))+o)
DEF FN teLong&(n,o)=PEEK LONG(PEEK LONG(TEHANDLE(n))+o)
:
LONG FN teLock&(n)
  texthndl&=FN teLong&(n,62)       :REM Get handle to text being edited
END FN = USR3(texthndl&)           :REM return pointer/locked mem block
:
:REM    Move Text from Edit Field to INDEX$(index)
:
:REM    On Entry:  field   Which EDIT FIELD in current window
:                  index   Which INDEX$ array to use
:REM    On Exit:   Returns # of Lines put in INDEX$
:
LONG FN EditToIndex(field,index)
  textPtr&=FN teLock&(field)           :REM Lock handle & return Point
  textLen =FN teWord(field,60):lines=0 :REM No. char's in text record
  CLEAR 0,index:CLEAR textLen+100,index:REM Clear & Make room
  temp$="" : textPtr&=FN teLock&(field):REM Initialize Stuff
  FOR  i = 0 TO textLen-1
    teChar = PEEK(textPtr&+i)          :REM Get char from text  block
    LONG IF teChar=13
      INDEX$(lines,index)=temp$
      temp$="" : lines=lines+1
    XELSE
      temp$=temp$+CHR$(teChar)         :REM add character to string
    ENDIF
  NEXT : INDEX$(lines,index)=temp$
  x=USR7(texthndl&)                    :REM Unlock Text Block
END FN = lines
:
:REM    Move INDEX$(index) to edit field 'field'
:
:REM    On Entry:  field   Which EDIT FIELD in current window
:                  index   Which INDEX$ array to use
:REM               lines   # of Lines to use out of INDEX$
:
```

continued...

MACINTOSH APPENDIX

TEHANDLE function continued...

```
LONG FN IndexToEdit(field,index,lines)
  L&=MEM(20+index) : H&=USR5(L&)      : IF H&=0 THEN BEEP : BEEP : RETURN
  T&=USR3(H&):A&=T&: I&=MEM(40+index)
  REM T& is temporary Ptr,     I& points to INDEX$
  FOR I=0 TO lines : sl%=PEEK(I&): I&=I&+1: REM get & Skip Length byte
    IF (T&-A&)+sl% => L& THEN PRINT"Program Error..": STOP
    BLOCKMOVE I&,T&,sl% :I&=I&+sl%  :T&=T&+sl%  :POKE T&,13 : T&=T&+1
  NEXT I : t&=FN teLong&(field,62): x=USR6(t&)  :REM dispose of old TEXT
  x=USR7(H&) : POKE LONG PEEK LONG(TEHANDLE(field))+62,H&
  POKE WORD PEEK LONG(TEHANDLE(field))+60,T&-A& :REM  SET TEXT LENGTH
  CALL TECALTEXT(TEHANDLE(field))
  CALL TEUPDATE(LargeRect,TEHANDLE(field))
END FN
:
:REM   Make Dummy Edit Field Text
:
FOR I=0 TO 5 : INDEX$(I)=HEX$(I)+STRING$(RND(99),"#") :NEXT
FN IndexToEdit(1,0,5)
:
:REM   Example uses of TEHANDLE
:
"WAIT LOOP"
  LOCATE 0,0
  PRINT "DEST RECT",FN teWord(1,0),FN teWord(1,2),
  PRINT FN teWord(1,4),FN teWord(1,6)
  PRINT "VIEW RECT",FN teWord(1,8),FN teWord(1,10),FN teWord(1,12),
  PRINT FN teWord(1,14)
  PRINT "SELECT PT",FN teWord(1,28),FN teWord(1,30),FN teWord(1,32),
  PRINT FN teWord(1,34)
  PRINT "TEXT LEN.",FN teWord(1,60),"lines=";FN teWord(1,94),
  PRINT "TEXT ADDR",PEEK LONG(FN teLong&(1,62))
  ON BREAK GOSUB "BREAK"
  BREAK ON
GOTO "WAIT LOOP"
BREAK OFF
:
:REM   Show ending contents of EDIT FIELD
:
"BREAK"
N=FN EditToIndex(1,0)
EDIT FIELD CLOSE 1 : CLS
FOR I=0 TO N
  PRINT INDEX$(I,0)
NEXT
STOP
```

REMARK See the TE toolbox calls in the "Toolbox" section of this appendix for other ways of manipulating the text in an EDIT FIELD.

MACINTOSH APPENDIX

TEXT statement

FORMAT **TEXT** *[font][, [point size], [face], [mode]]*

DEFINITION This statement is used to change the characteristics of the text going to the printer and to the screen. While you may also use the text commands of quickdraw, ZBasic provides this easy function to change text characteristics with one statement instead of four. Parameter values:

#	FONT	#	FACE	#	MODES
0	System font	0	Plain	0	Source COPY
1	Application font	1	Bold	1	Source OR
2	New York	2	Italic	2	Source XOR
3	Geneva	4	Underlined	3	Source BIC
4	Monaco	8	Outlined	4	NOT source COPY
5	Venice	16	Shadow	5	NOT source OR
6	London	32	Condensed	6	NOT source XOR
7	Athens	64	Extended	7	NOT source BIC
8	San Francisco	(19=bold,italic and shadow)			
9	Toronto				
10	Seattle				
11	Cairo	POINT SIZE			
12	LosAngleles	1 through 127			

Common laser fonts: (20-Times, 21-Helvetica, 22-Courier, 23-Symbol, 24-Taliesin)

Default TEXT attributes: `TEXT 4,9,0,1`

FONT: Monaco
SIZE: 9
FACE: 0
MODE: 1

NOTE: TEXT affects output to both the screen AND PRINTER (Imagewriter or Laserwriter). LPRINT is normally used for text output to the printer. If you wish to ROUTE both TEXT and Graphics output to printer use ROUTE 128.

To determine what fonts are available on your system, use this routine:

EXAMPLE

```
REM Determine the fonts and numbers on current system
FOR X = Ø TO 255
  CALL GETFONTNAME(X, A$)
  IF LEN(A$) THEN PRINT X, A$
NEXT
```

REMARK Most Macintosh fonts are proportionally spaced and printing columns of data is different from other computers. The MONACO font is an excellent one for mono-spaced characters to simulate other systems.

64K ROM BUG NOTE: The original 64K Macintosh™ ROMS have a bug in the source copy mode and print a few extra spaces after the text. Therefore ZBasic sets the text mode to Source OR. To change it to print over the background completely, use TEXT,,,Ø at the beginning of your program. This bug was fixed in the 128K ROMS (like the Macintosh Plus™).

MACINTOSH APPENDIX

TIMER function

FORMAT TIMER

DEFINITION Returns the number of seconds that have elapsed since the internal clock was at midnight. For instance, 12 noon would return 43,200. There are 86,400 seconds in a day.

EXAMPLE

```
start&=TIMER
PRINT "Start time=";start&
:
DELAY 10000
:
endtime&=TIMER
PRINT "End time=";endtime&
:
PRINT" Total elapsed time=";endtime&-start&
END

RUN

Start time= 2311233
End time= 23311243
Total elapsed time= 10
```

REMARK Also see TIMER ON, TIMER OFF, ON TIMER GOSUB and TIME$.

Note: TIMER will always return the correct count whether or not the TIMER ON function has been enabled.

MACINTOSH APPENDIX

TIMER statement

FORMAT

TIMER ON
TIMER OFF
TIMER STOP

DEFINITION TIMER is used to create a TIMER event every *n* seconds. In other words; the program will be routed to the subroutine specified by ON TIMER *n* GOSUB every *n* seconds when physically between TIMER ON and TIMER OFF statements.

TIMER ON	Enables the timer event trapping and calls a routine you specify with ON TIMER(n) GOSUB suboutine. *n* is a number between 1 and 86,400 which specifies the number of seconds between TIMER events.
TIMER OFF	Disables TIMER ON (TIMER function still works)
TIMER STOP	Temporarily disables the timer event checking and stores events in a queue until TIMER ON is executed again.

EXAMPLE

```
WINDOW#1
MENU ON: TIMER ON:BREAK ON
ON TIMER 10 GOSUB "Timer Event"
ON MENU GOSUB "Menu Event"
:
MENU 1, 1, 1, "/QQuit"
:
"Main Event Loop"
GOTO "Main Event Loop"
MENU OFF:TIMER OFF
:
"Menu Event"
X=MENU(0):Y=MENU(1)
IF X=1 AND Y=1 THEN END
RETURN
:
"Timer Event"
J&=TIMER
PRINT "LAST TIMER";I&
PRINT "THIS TIMER";J&
PRINT "--------------------------"
PRINT "TIME ELAPSED";J&-I&
I&=J&
RETURN
```

REMARK See TIMER function, TIME$ and ON TIMER GOSUB for more information.

Note: The TIMER function will always return the correct count whether or not the TIMER ON function has been enabled.

MACINTOSH APPENDIX

TRON #128 statement

FORMAT **TRON#128**

DEFINITION This statement is an enhancement to the standard TRON that routes all line number trace output to the printer. Great for debugging without disturbing screen output.

EXAMPLE

```
TRON#128
FOR X=1 TO 2
NEXT
END

RUN
```

Printer output:

```
 00001 00002 00003 00002 00003 00004
```

REMARK See TRON in the main reference manual for more information. Also see TRON V and TRON MONITOR in this appendix.

MACINTOSH APPENDIX

TRON statement

FORMAT **TRON** =*variable name*
TRON V

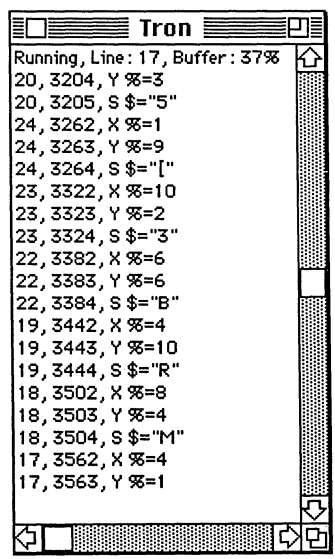
DEFINITION Same as the standard ZBasic TRON with the following enhancements:

TRON=variable	Sets up the variables whose names and values will be passed to the TRON MONITOR desk accessory when TRON V is enabled.
TRON V	Sets up variable value passing events to work with the ZBasic TRON MONTOR desk accessory (included with ZBasic).

EXAMPLE

```
REM Enable the TRON MONITOR Desk accessory
REM BEFORE executing the program or use:
REM X%=FN OPENDESKACC("Tron Monitor") See DA example in Toolbox.
WINDOW#1, "EXAMPLE OF TRON V AND TRON=VAR"
PRINT"Be sure to "CLICK" the Tron monitor DA window and "Watch"
PRINT:PRINT"Since Single-Step is on you will need to press a key"
PRINT"to advance to the next line statement.
DIALOG ON:BREAK ON
TRON V: REM Enable trace window event trapping
REM   Check the Variables S$, X and Y
TRON=S$:TRON=X: TRON=Y
DO
  X=RND(10):Y=RND(10)
  S$=CHR$(RND(64)+33)
UNTIL LEN(INKEY$)
END
```

OUTPUT:



REMARK See TRON MONITOR DA information on next page.

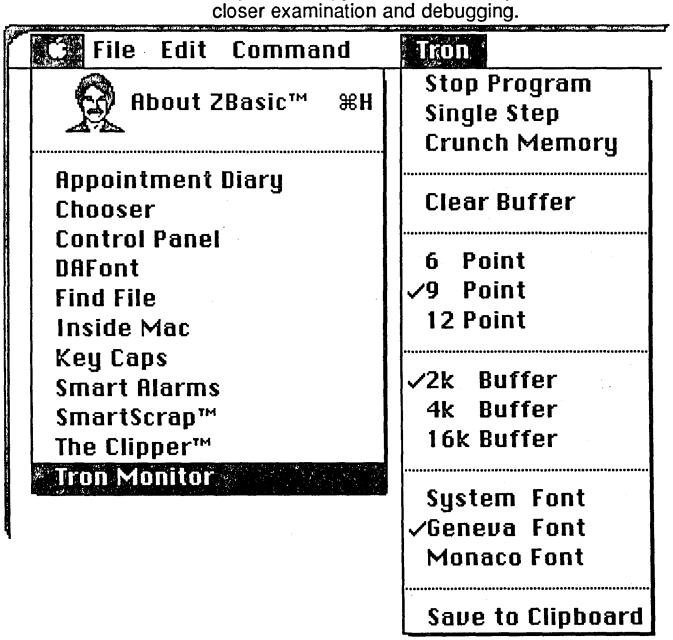
MACINTOSH APPENDIX

TRON MONITOR Desk Accessory

FORMAT

DEFINITION This desk accessory comes with ZBasic and is installed using FONT/DA Mover. The TRON MONITOR is activated from the menu and values of variables selected with TRON=variable are shown in the TRON MONITOR window.

Stop Program	May be used to exit the program instead of <COMMAND PERIOD>
Single Step	The same as TRON S
Crunch Memory	Same as doing MEM(-1)
Clear Buffer	Erases the contents of the Tron Buffer
6,9 or 12Point	Sets the point size of the text in the window
2, 4 or 16K Buffer	Sets the buffer size
Font	Sets the font used for text in the Tron Monitor window
Save to Clipboard	Saves the contents of the Tron Monitor window to the clipboard so you can copy them into a word processor or ZBasic's editor for closer examination and debugging.



EXAMPLE See TRON V on previous page.

REMARK The TRON window must be activated in stand-alone applications when regualr event trapping is being used. Without event trapping acitvated the menu cannot be pulled down and TRON MONITOR selected.

MACINTOSH APPENDIX

USR defaults

FORMAT *Variable&*=**USRn**(*expr*)

DEFINITION Pass *expr* to a USR routine defined by ***n*** and return value in *Variable&*.

REMARK These USR functions are the defaults and contain many powerful tools for the experienced and not-so-experienced programmer.

USR0 Allocates Non-Relocatable Memory Block on the Heap

Toolbox	See *_NewPtr* in "Inside Macintosh"
expr	Number of bytes to Allocate
Variable&	Returns the start address in memory. If Value = 0 the block was not allocated (error).

Example

```
X&=USR0(10000)
LONG IF X&=0
  PRINT"Not Enough memory!":STOP
XELSE
  PRINT PEEK(X&) :REM  Points to first of 10000 bytes
END IF
```

USR1 Releases Non-relocatable Memory Block from Heap

Toolbox	See *_DisposPtr* in "Inside Macintosh".
expr	The LongInteger pointer to the Block you want to release
Variable&	Returns the address of the Block

De-allocates the memory block pointed to by 'expr' allowing the memory to be used by other blocks

Example
```
X&=USR1(X&)
```
Warning If X& does not point to a memory block expect a system error!

USR2 High-Speed LongInteger Square Root

expr	LongInteger number
Variable&	Returns LongInteger Square root of *expr*

Example
```
X&=USR2(144)
```
(Returns 12)

continued...

MACINTOSH APPENDIX

USR functions continued...

USR3 Locks a Relocatable Memory block (keeps from moving in memory)

Toolbox — See *_Hlock* "Inside Macintosh".
expr — Handle to relocatable memory block
Variable& — Returns the Address to the start of the Locked block

Note: Only pass a LongInteger (do not use a 16 bit integer).

Example

```
X&=USR5(10000)      : REM Allocates a Handle to a relocatable block
Y&=USR3(X&)         : REM Locks the block and returns Pointer
POKE Y&,123         : REM Puts 123 to the 1st byte of block
Y&=USR7(X&)         : REM Un-Locks the block (Still avail)
GOSUB"far away"     : REM May cause Memory block to move
Y&=USR3(X&)         : REM Locks the block and returns Pointer
Z=PEEK(Y&)          : REM Gets the 1st byte of block
Y&=USR6(X&)         : REM Disposes of the memory block
```

USR4 Returns the address of a File I/O Paramblock

expr — ZBasic file number
Variable& — Returns a pointer to the I/O Paramblock.

Example — X&=USR4(2)

USR5 Allocate a Relocatable Memory Block

Toolbox — See *_NewHandle* in "Inside Macintosh".
expr — Size of the Block requested in Bytes (LongWord if >32,767)
Variable& — Returns the Handle to the Relocatable block
If Value=0 then it could not allocate the block!

Example — See USR3 example

USR6 Dispose of a Relocatable Memory Block pointed to by the Handle.

Toolbox — See *_DisposHandle* in "Inside Macintosh".
expr — Handle to the memory block to be Disposed (LongInteger)
Variable& — Unknown

Example — See USR3 example

USR7 Unlock a "Locked" Handle

Toolbox — See *_HUnlock* in "Inside Macintosh".
expr — Handle to Locked Relocatable memory block (LongInteger)
Variable& — Unknown

Note — USR6 automatically Un-locks the block first.

USR8 See USR8 in the main reference section (INTEGER SINE)
USR9 See USR8 in the main reference section (INTEGER COSINE)

MACINTOSH APPENDIX

WAVE statement

FORMAT **WAVE** *voice* [, {*SIN* | *var%(n)* }]

DEFINITION Sets the WAVE form of the sound output used with the SOUND statement.

voice	Specifies one of the Macinbtosh voices (0-3). Default is zero.
SIN	Specifies a Sine Wave (this is the default)
var%(n)	Sets the WAVE format in the an integer variable array. There must be at least 256 elements in the array.

EXAMPLE

```
DIM A%(255) : CLS
FOR I=0 TO 255 : A%(I)=RND(32)+I/4 : NEXT
WAVE 3,A%(0) : WAVE 0
PRINT"RANDOM WAVE SOUNDS"
FOR I=100 TO 800 STEP I
  SOUND I,500
NEXT
PRINT"4 VOICE SOUNDS"
FOR I=1 TO 8
  WAVE 1 : SOUND WAIT
  FOR J=0 TO 3
    READ F : SOUND F,400,,J : PRINTF,
  NEXT
  SOUND RESUME : PRINT
NEXT
DATA 400,400,400,0
DATA 400,400,400,0
DATA 400,800,800,0
DATA 500,500,100,200
DATA 100,200,400,800
DATA 100,110,120,130
DATA 400,450,500,550
DATA 0,0,0,400
PEN ,,,10
DO
  FOR I=0 TO 255 STEP 4
    CIRCLE FILL 512,384+192,192 TO I,4
  NEXT
  TRONX
UNTIL SOUND=0
```

REMARK Also see SOUND function and SOUND statement.

MACINTOSH APPENDIX

WIDTH statement

FORMAT

WIDTH *-n*
WIDTH LPRINT *-n*

DEFINITION

Disable automatic wrap of text. This can speed up printing by a factor of 4-10 times.

WIDTH-2	Disables text wrap-around in the screen.
WIDTH LPRINT -2	Disables text wrap-around to the printer.
WIDTH -1	Sets back to normal wrap-around mode.
WIDTH LPRINT -1	Sets back to normal wrap-around mode.

EXAMPLE

```
WINDOW#1, "DEF LPRINT":WIDTH LPRINT-2
DEF LPRINT
IF PRCANCEL THEN PRINT"Printing Aborted":STOP
:
ROUTE 128
PRINT"HELLO"
CIRCLE FILL 200,200,200
CLEAR LPRINT
ROUTE 0
END
```

REMARK

Also see WIDTH in the main reference section for other options.

MACINTOSH APPENDIX

WINDOW function

FORMAT **WINDOW** (*expression*)

DEFINITION The WINDOW function returns important information about the active and/or output window being used:

GET ACTIVE AND OUTPUT WINDOW

WINDOW(Ø) Returns the window number of the current active window. WINDOW(Ø) returns Ø if there is no current active window.

WINDOW(1) Returns the window number of the current output window. All printing, graphics, and quickdraw graphics are directed to this window (this will normally be the active window unless WINDOW OUTPUT has been used, in which case the Active and Output windows may be different).

ZBasic allows up to 15 windows on the screen at the same time (memory permitting). The forward-most highlighted window is the "active window" and is the window where all activity takes place unless ZBasic is directed to focus activity on another window with the WINDOW OUTPUT statement.

The WINDOW statement will make the window with that window number the active window and bring it to the front. The "active window" and "output window" are the same unless the WINDOW OUTPUT statement is used to direct output to another window other than the "active window". This may be done for re-drawing background windows or for other updating.

The current output window is where the BUTTON, EDIT FIELD and DIALOG functions return information.

Sending output to another window other than the "active window" is normally done when a window refresh is called for, such as after a dialog window erases part of a background window. The only time you do not have to replace window contents is when a menu pull-down is done (see GETPUT.BAS and WINDOWPIC.BAS examples on the master disk).

GET WINDOW SIZE IN PIXELS

WINDOW(2) Returns the width of the current output window.
WINDOW(3) Returns the height of the current output window.
WINDOW(4) Returns the width position in the current output window where the next character will be printed.
WINDOW(5) Returns the height position in the current output window where the next character will be printed.

continued...

WINDOW function continued...

GET WINDOW SIZE IN RELATIVE COORDINATES

WINDOW (6)	Returns the width of the current output window.
WINDOW (7)	Returns the height of the current output window.
WINDOW (8)	Returns the width position of the current output window where the next character will be printed.
WINDOW (9)	Returns the height position of the current output window where the next character will be printed.

GET WINDOW POSITION ON THE SCREEN (Pixel Coordinates (screen))

WINDOW (10)	Returns the horizontal pixel position of the upper-left-hand-corner of the window on the screen.
WINDOW (11)	Returns the vertical pixel position of the upper-left-hand-corner of the window on the screen.
WINDOW (12)	Returns the horizontal pixel position of the lower-left-hand-corner of the window on the screen.
WINDOW (13)	Returns the vertical pixel position of the lower-left-hand-corner of the window on the screen.

GET WINDOW GRAFPORT POINTER

WINDOW (14) Returns a pointer& to the grafport of the window for use with toolbox calls (same as GET WINDOW).

Example: WPTR&=WINDOW(14)

Also see CALL GETPORT and GrafPort in the toolbox chapter.

REMARK * Also see COORDINATE, DIALOG function, WINDOW statements and GET WINDOW.

MACINTOSH APPENDIX

WINDOW statement

FORMAT

WINDOW [#]*window number*[, [*title$*[, [(*x1,y1*)-(*x2,y2*)][, *type*]]]
WINDOW CLOSE [#] window number
WINDOW OFF
WINDOW OUTPUT [#]*window number*

DEFINITION

The ZBasic Window statements and functions are very powerful and will let you take full control of the Macintosh windowing environment.

There are many possible ways of configuring windows. Parameter definitions:

window number — The number of the window being referred to (expression between 1 and 15). Window numbers outside this range will "wrap-around" i.e. number 17 will converted to window one etc.

title$ — A title that appears in the Title bar at the top of all window types except 2,3 and 4. The title will be centered. System font is used.

(x1,y1)-(x2,y2) — The window size is given in actual pixel coordinates (NOT ZBasic coordinates), with x1 and y1 as the upper left hand corner and x2 and y2 as the lower right corner . Default size is (4,38)-(510,320) (which is approx. 80x25 lines in Monaco 9 point text).
The smallest window size is 64x64 pixels.
The largest window size is 8192x8192 pixels (for large displays).

type

1 Document window. This window has a title bar at the top and a size box at the bottom right hand corner so the user may size the window.

2 Dialog window with a border of double lines. This window has no other attributes and may not be moved or sized. The user will normally just "click" the window to make it go away. This Dialog window is used for giving error messages or other messages to the user.

3 Window with a simple one line border. May be used similar to number 2 above.

4 Window with a shadow. Same as number 3 but with a shadow.

5 User may not change window size. Same as WINDOW type 1 but without a grow box and gives a slightly larger view area than window type 1.

6-8 Undefined.

9 Adds a ZOOM BOX to the title bar in HFS. This is ignored with MFS. Example: `WINDOW #1, "Example",,9`. Type 9 and 265 are the only types with Zoom. See DIALOG 8 and 9 for ways of determining when the ZOOM box events occur. Same as window type one in other ways.

* Subtract one to get the Toolbox window ID (see Window in Toolbox section).

continued...

MACINTOSH APPENDIX

WINDOW statement continued...

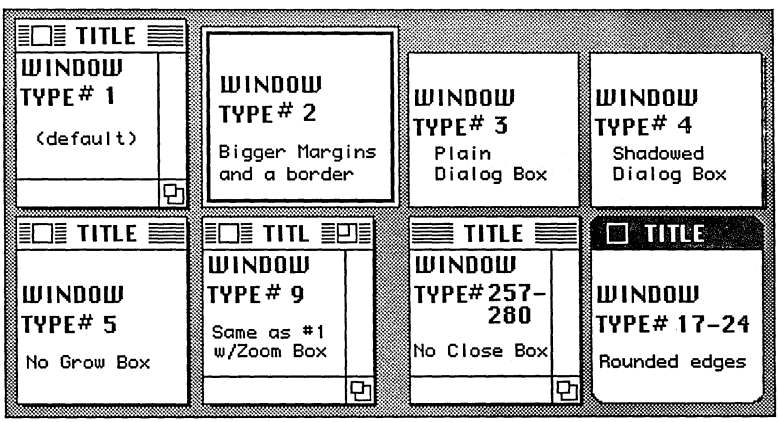
Other Window Types — The windows may be modified by making the window type negative or by adding certain values to it:

Negative type — MODAL DIALOG WINDOWS. The user cannot click outside the window or use MENUs. Example: `WINDOW #1, "Example",,-1`

Add 256 to type — DISABLE GO-AWAY BOX for that Window:
Example: `WINDOW #1, "Example",,257`

Types 17-24 — ROUNDED CORNERS. 18=least rounding----24=most rounding.
Example: `WINDOW #1, "Example",,17`

WINDOW TYPES



NOTE: To activate window selection and window events be sure to use event trapping like ON DIALOG. Also see GET WINDOW, WINDOW function and COORDINATE.

FORMAT **WINDOW CLOSE** *[#]window number*

DEFINITION Removes the window specified by *window number* from the screen.

It is important that there always be a window to direct graphics and printing. Therefore you should redirect output to another window with WINDOW OUTPUT#n or WINDOW#n before closing the active window.

continued...

MACINTOSH APPENDIX

WINDOW statements continued

FORMAT **WINDOW OFF**

DEFINITION Disables the ZBasic default command window so that it does not appear in Stand-Alone, double-click applications. It will always appear during interactive programming.

FORMAT **WINDOW OUTPUT** *[#] window number*

DEFINITION Directs output to this window (not necessarily the active window).

The Active Window is normally the current selected window where all graphics and printing are directed. There may arise a situation where you will wish to direct output to another window without de-activating the current active window.

Use this statement to accomplish that. Use this statement (or WINDOW#n) again to re-direct output back to the active window.

FORMAT **WINDOW** *[#] window number*

DEFINITION Makes the window specified by *window number* the ACTIVE window. This brings the window to the front. If no window is currently open, a window will be created (default is window type 1 with a size of (4,38)-(510,320).

EXAMPLE

```
CLS: Y=50
DIALOG ON
FOR Type=1 TO 5: IF Type>3 THEN Y=200
  IF Type<4 THEN X1=(Type-1)*150 +25ELSE X1=(Type-3)*150 +25
  Title$="Window"+STR$(Type)
  :
  WINDOW #Type, Title$, (X1, Y)-(X1+100, Y+100), Type
  PRINT@(,5);"Window type";Type
  PLOT 0,0 TO 1023, 768
  INPUT"Press <ENTER>";A$
NEXT
DIALOG OFF
FOR X=1TO5
  WINDOW #X: REM   Makes the WINDOW ACTIVE
  INPUT "<ENTER to CLOSE WINDOW";A$
WINDOW CLOSE #X
NEXT
END
```

REMARK See IOTEST.BAS on the master disk for more examples of creating your own window types.

MACINTOSH APPENDIX

WINDOW PICTURE statement

FORMAT **WINDOW PICTURE** [#] *window number, PictureHandle&*

DEFINITION Sends a picture handle to the Macintosh Window Manager for use to refresh the specified window contents should the window be overwritten with graphics, Desk Accessories or other windows (update events will not be generated).

window number	The window to be used.
PictureHandle&	A LongInteger variable used as a handle for a PICTURE. See the PICTURE function and statement for details.

EXAMPLE

```
WINDOW OFF : WINDOW#1,,,4
:
REM  Create a PICTURE for use with WINDOW PICTURE
PICTURE ON
  PRINT"THIS IS A TEST"
  CIRCLE 512,384,384
  PLOT 1023,0 TO 0,767
PICTURE OFF,PIC&
:
REM Display PICTURE specified by PIC&.
PICTURE, PIC&
:
REM Set PIC& for refresh. Try opening DA's
WINDOW PICTURE #1,PIC&
:
ON BREAK GOSUB "STOP UPDATE": BREAK ON
:
"Main Event Loop"
GOTO "Main Event Loop"
:
"STOP UPDATE"
WINDOW PICTURE #1,0: REM Stop Updating window
KILL PICTURE PIC&: REM  Delate PICTURE from memory
END
```

REMARK Also see PICTURE, KILL PICTURE, GET and PUT graphics and WINDOW. Also see FN GETPICTURE in the Toolbox section of this appendix.

MACINTOSH APPENDIX

WRITE FILE statement

FORMAT **WRITE FILE** [*#*] *filenumber%, destinationAddress&, NumberofBytes* [*&*]

DEFINITION Allows high-speed disk write. Used in conjunction with READ FILE#.

destinationAddress&	LongInteger address where writing begins.
NumberofBytes[&]	Bytes to write.

EXAMPLE1

```
REM This program allows you to Execute other applications
REM and return with all variables as they were
REM THIS APPLICATION MUST BE IN SYSTEM FOLDER TO WORK WITH HFS
REM NOTE: ALL VARIABLES ARE RESTORED UPON RETURN TO THIS PROGRAM
:      :
WINDOW#1,"Execute other Applications and return"
FIRST%=0: GOSUB"Get Last Variable pointer"
REM Check to see if returning from another application
OPEN"R",1,"VARS"          REM    If returning from an application
LONG IF LOF(1,1)>10       REM    reload all the variables
  READ FILE #1,VARPTR(FIRST%), LAST&-VARPTR(FIRST%)
  CLS: BEEP:BEEP:TEXT 1,24,1
  PRINT:PRINT"Returning from: ":PRINTAPPL$
  PRINT:PRINT"Press a key..."
  DO:UNTIL LEN(INKEY$)
END IF
:
REM *** EXIT TO MAC PROGRAM AND RETURN TO THIS PROGRAM ***
REM Moves This Applications filename to FINDER name
FOR I=0 TO 15: REM Your filename MUST be less than 16 char's
  POKE &2E0+I,PEEK(&910+I)
NEXT
REM *** Get name of application to execute
APPL$=FILES$(1,"APPL",,Volume%):  REM APPL=any executable file
LONG IF APPL$=""
  CLOSE#1
  KILL "VARS"
  REM  Restore FINDER filename as desktop start-up
  A$="FINDER"
  FOR I=0 TO LEN(A$)+1
    POKE&2E0+I,PEEK(VARPTR(A$)+I)
  NEXT
XELSE
  RECORD#1,0,0: REM  Reset file pointer to beginning of file
  REM *** Save Variables before executing Application
  WRITE FILE #1,VARPTR(FIRST%), LAST&-VARPTR(FIRST%)
  CLOSE#1
  REM Execute Application now...
  RUN APPL$,Volume%
END IF
END
"Get Last Variable pointer"
LAST&=VARPTR(LAST%):RETURN
REM Last variable used as a dummy for READ FILE and WRITE FILE
```

REMARK Also see READ FILE#.

MACINTOSH APPENDIX

MACINTOSH
TOOLBOX

MACINTOSH APPENDIX

MACINTOSH TOOLBOX

The Macintosh comes with many powerful, built-in functions and statements. ZBasic gives you access to most of them. While the Mac ROM calls and functions are indeed useful, the technical "Inside Macintosh" was written in Hieroglyphics as far as most BASIC programmers are concerned.

This section of the ZBasic manual attempts to simplify and enlighten BASIC programmers to the Macintosh toolbox routines. While there are a number of examples, definitions and illustrations, there may be unpleasant occasions when you will have to refer to "Inside Macintosh" (it is 1500+ pages). Hopefully this section of the manual will keep the need for this to a minimum.

WHAT WE DON'T INCLUDE

Operating System and Memory Manager traps are excluded since they are both register based instead of stack based and would conflict with the internal workings of ZBasic.

GLOSSARY OF TERMS AND DEFINITIONS USED THROUGHOUT THE TOOLBOX

The toolbox refers to many terms that may be foreign to you. The following paragraphs discuss some of these terms and will be used throughout this section.

There are also special terms discussed at the beginning of each section (like QuickDraw and Resource Manager) that are specific to those toolbox types. Be sure to study those glossaries as well. The terms used will be certain data or variable types as defined below:

DATA TYPES

NUMERIC

The numeric terms under "Type"; Byte, Word, Long and Fixp, will refer to the following ranges and memory requirements.

--Type --	--- Bits ---	---Bytes---	---- Positive ----	---- Signed -----
Byte	8 bit	1	0 to 255	-128 to +127
Word	16 bit	2	0 to 65,535	-32,768 to +32,767
Long	32 bit	4	0 to 4,294,967,294	±2,147,483,647
Fixp	32 bit	4	-- None--	±32,767.999,9

BOOLEAN

Byte	8 Bit	Boolean	True <> 0	False = 0

STRING

Char	Byte	Passed as Word with ASCII in both hi & low Bytes
Str	Len+String	Byte length followed by up to 255 characters (either a quoted string or a string variable).

EXPRESSIONS

Wherever Byte, Word, Long, Fixp or Char are used in toolbox functions or calls you may substitute expressions (not str).

This means you may use a number, variable or numeric expression in place of that type and values exceeding the range specified will be truncated.

Important Note: When you see a variable specifier like %, &, $ or Var in a toolbox function or call, you MUST use a variable there!

MACINTOSH APPENDIX 

VARIABLE TYPES

When you encounter one of the variable type specifiers; %, &, $ or Var, it means you MUST use that variable type. The routine uses the pointer to that variable (VARPTR) and not the value stored in the variable (at least not directly).

-Variable-	--Type---	
%	Integer	Pointer to Integer Variable
&	Long Integer	Pointer to Long Variable
$	String	Pointer to String Variable
Var	Any Variable	Pointer to Any Variable

Important Note: when you see a variable specifier always use that variable type!

WHEN TO USE EXPRESSIONS OR VARIABLES

This toolbox chapter contains terms with a number of definitions that begin with a name. If the name includes a variable specifier than you must use a variable. If no specifier is given and a type follows like; Byte, Word, Long, Str , Fixp, or Char. Use that type:

Examples:

Cnt	Word	Count Word Value (0 to 32,767 typically)
Name	Str	Any Name "quotes" or Variable$
Name$	Var$	A string variable only (specified by $)
Ptr	Long	An expression, number or numeric variable
Ptr&	Long	A LongInteger variable only (specified by &)

POINTER TO MACHINE LANGUAGE ROUTINES

Used to point to the address of a user defined machine language routine. In most cases this should be defined as zero.

Proc	Long	Pointer to a Machine Language Routine

'POINT'

Some routines, primarily QuickDraw, List Manager and some MOUSE functions use what we'll refer to as *Point*. This signifies that the four bytes starting at the position of a variable are significant. These bytes can be accessed using a number of different variables:

Point	Var	Var$, Var& or Var%,Var%

Point Does: PEEK LONG(VARPTR(Var)) or POKE LONG(VARPTR((Var))

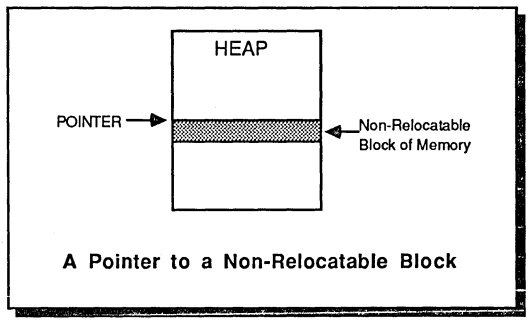
An example of using Point would be to return the position of the MOUSE where the first two bytes return the vertical position and the last two bytes return the horizontal position.

MACINTOSH APPENDIX

POINTERS (Ptr)

A Pointer is the Long address of the actual byte(s) to be used. Pointers never move in memory (non-relocatable).

Ptr	Long	FirstByte=	PEEK(Ptr)
		SecondByte=	PEEK(Ptr+1)



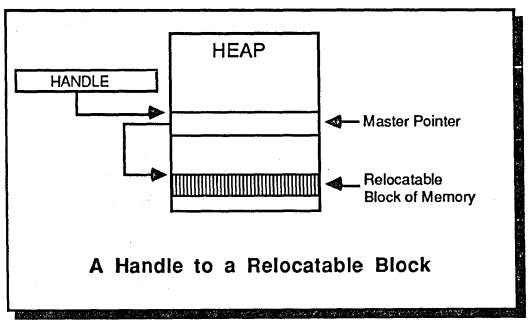
A Pointer to a Non-Relocatable Block

HANDLES (Hndl)

A Handle is used to point at the current position of a relocatable pointer in memory. This position may be moved according to the condition of memory and the whims of the Macintosh Memory Manager.

Hndl	Long	FirstByte=	PEEK(PEEK LONG(Hndl))
		SecondByte=	PEEK(PEEK LONG(Hndl)+1)

Handles may be locked and unlocked with the ZBasic USR3 and USR7 statements respectively. Don't leave a Handle locked for too long (see USR 3 and 7 in this appendix for more details of usage).



A Handle to a Relocatable Block

Handles are referred to quite frequently in the toolbox. Relocatable blocks of memory are used so the Macintosh Memory manager can make the most efficient use of memory.

ZBasic compiled programs take advantage of the Memory Manager by creating segments when compiling. See SEGMENT and SEGMENT RETURN for ways of controlling the memory segments of your ZBasic programs.

MACINTOSH APPENDIX 

Resource Manager

See some examples of Resource Manager under "PICTURE" and "Clipboard" in this appendix.

The following terms are used with the Resource Manager routines:

Type	Long	Example: DEFSTR LONG : T&=CVI("TEXT")
Rsize	Long	Resource size in bytes
Indx	Word	Index number (From 1 to n typically)
ID	Word	ID number of resource
Attr	Word	File attributes (See "Inside Mac")
RefNum	Word	Resource file reference number
ErrNum	Word	Resource manager error number
Fname	Str	File name: "quoted string" or Variable$
Fname$	Var$	File name in variable$
Rhndl	Long	Handle to resource

Returns	Functions----------	Definitions----------
Word%=	FN INITRESOURCES	*** Conflicts with ZBasic Internally! ***
RefNum%=	FN OPENRESFILE (Fname)	Opens existing Resource file by name
ErrNum%=	FN RESERROR	Returns last resource manager error
RefNum%=	FN CURRESFILE	Returns current Refnum of resource file
RefNum%=	FN HOMERESFILE (Rhndl)	Returns Refnum of file with ResHandle
Cnt%=	FN COUNTTYPES	Returns Total number of Resource type(s)
Cnt%=	FN COUNTRESOURCES (Type)	Returns the number of Resources of Type
Rhndl&=	FN GETINDRESOURCE (Type,Indx)	Get Handle to Resources using Type & Index
Hndl&=	FN GETRESOURCE (Type,ID)	Get Handle to Resource using Type & I.D. #
Rhndl&=	FN GETNAMEDRESOURCE (Type,Name)	Get Handle to Resource using Type & Name
ID%=	FN UNIQUEID (Type)	Returns an ID% for Type not yet used
Rsize&=	FN SIZERESOURCE (Rhndl)	Returns sze in bytes for Resource Handle
Attr%=	FN GETRESFILEATTRS (RefNum)	Returns file attributes

	Procedures----------	Definitions----------
CALL	RSRCZONEINIT	*** Conflicts with ZBasic Internally! ***
CALL	CREATERESFILE (Fname)	Create a new Resource file called Fname
CALL	CLOSERESFILE (RefNum)	Close Resource file specified by RefNum
CALL	USERESFILE (RefNum)	RefNum Resource file becomes first searched
CALL	GETINDTYPE (Type&, Indx)	Indx=1 to Cnt%(above): Returns types in Type&
CALL	SETRESLOAD (Byte)	If Byte=0 then won't load --Non-Zero will load
CALL	LOADRESOURCE (Rhndl)	Reads Resource pointed to by Rhndl into Memory
CALL	RELEASERESOURCE (Rhndl)	Release Resource pointed to by Rhndl from Memory
CALL	DETACHRESOURCE (Rhndl)	Makes hndl Not Part Of Resource
CALL	GETRESINFO (Rhndl,ID%,Type&,Name$)	Returns ID,Type,Name of Rhndl
CALL	SETRESINFO (Rhndl,ID,Name)	Changes The ID Number & Name of Rhndl
CALL	SETRESATTRS (Rhndl,Attr)	Changes Attributes to Attr of Rhndl
CALL	CHANGEDRESOURCE (Hndl)	Set Resource in memory changed (to write).
CALL	ADDRESOURCE (Rhndl, Type, ID, Name)	Adds resource (Rhndl) of Data, Res Type, id, Name
CALL	RMVERESOURCE (Rhndl)	Remove Rhndl from ResFile
CALL	UPDATERESFILE (RefNum)	Update resource file on disk with memory
CALL	WRITERESOURCE (Rhndl)	Writes Resource Data specified by Rhndl to Disk
CALL	SETRESPURGE (Byte)	Allows modifing purgable Resource to 0 or -1
CALL	SETRESFILEATTRS (RefNum, Attr)	Sets the ResFile of RefNum Attributes

-------128 ROMS ONLY----------		Definitions----------
Rsize=	FN MAXSIZERSRC(Hndl)	Returns the Maximum Resource size
Roff=	FN RSRCMAPENTRY(Hndl)	Returns Offset (Roff) in Resource Map
Word=	FN OPENRFPERM(Fname,Vref,Perm)	Open Resource file, Fname, Vref=VrefNum Perm=Permission Read/Write etc...

MACINTOSH APPENDIX

QUICKDRAW GRAPHIC ROUTINES

QuickDraw is a collection of powerful graphic tools built into the Macintosh ROM.

Note that ZBasic utilizes these routines in most of it's graphic statements and functions like; PLOT, CIRCLE, BOX, COLOR, PEN, GET, PUT, PICTURE so that you will not need to do QucikDraw routines for these. These routines will come in handy for inverting, Regions, Polygons etc.

The following terms are used with the QuickDraw routines:

POINT AND RECT

Point	Var	Pointer to the first byte of two "Words" specifying Y, X
Rect	Var	Pointer to the first byte of four "Words" specifying Y, X and Y2, X2 that are the top-left and bottom-right coordinates of a rectangle.
SRect	Rect	Defines the source Rectangle (same structure as Rect)
DRect	Rect	Defines the target Rectangle (same structure as Rect)

Examples of ZBasic variables with Point and Rect

Variable Order	Rect Variable	Point1	Point2	Top (Y)	Left(X)	Bottom(Y)	Right(X)
DIM A%(3)	A%(0)	A%(0)	A%(2)	A%(0)	A%(1)	A%(2)	A%(3)
DIM T,L,B,R	T	T	B	T	L	B	R
DIM TL&,BR&	TL&	TL&	BR&	TL& *	TL&**	BR&*	BR&**
DIM P2&(1)	P2&(0)	P2&(0)	P2&(1)	P2&(0)*	P2&(0)**	P2&(1) *	P2&(1)**

*The High byte. **The low byte.
Note: In most cases we use the T,L,B,R format in our example programs.

Coord	Size	Offset	Offset	---- Range ----
Top	Word	Rect+0	Point1+0	-32767 to 32767
Left	Word	Rect+2	Point1+2	-32767 to 32767
Bottom	Word	Rect+4	Point2+0	-32767 to 32767
Right	Word	Rect+6	Point2+2	-32767 to 32767

Note: The coordinates are in the reverse order to ZBasic coordinates (Y, X instead of X, Y).

OTHER COMMON QUICKDRAW TERMS

globe	Long	Long Word Pointer to application globals (see A5 Memory next page)
GrfPtr	Long	Pointer to space for GrafPort (usually inside current window)
TxtPtr	Long	Pointer to text in memory (points to first character not length byte)
Spos	Word	Start offset position in bytes
RgnHnd	Long	Handle to a Region
BitMap	Long	Pointer to BitMap information
xSize	Word	Horizontal size in pixels
ySize	Word	Vertical size in pixels
xPos	Word	Left edge of horizontal position
yPos	Word	Right edge of horizontal position
dx	Word	Delta X Position (add that to current position to get new position)
dy	Word	Delta Y Position (add that to current position to get new position)
Angle	Word	Degrees (Integer 0 to 360)

MACINTOSH APPENDIX 

PATTERNS, CURSOR and PEN TERMS

Pat	Var	Pattern Definition 8 bytes (Bits:1=Black)
PHndl	Long	Handle to pattern of 8 Bytes
CHndl	Long	Handle to cursor structure
Curs	Var	Cursor pattern: 68 total bytes
Offset	Type	Description
0	Word16	Cursor shape
32	Word16	Cursor mask
64	Point	Hotspot
PenRec	Var	PenState Record 18 bytes
Offset	Type	Description
0	Point	PnLoc (Pen location)
4	Long	PnSize (Pen size)
8	Word	PnMode (Pen mode)
10	Byte8	Patttern
Smode	Word	Source Transfer Mode
Pmode	Word	Pattern Transfer Mode
Color	Long	Color to be used (Original QuickDraw. QuickDraw II will vary)

Smode	Mode	Pmode	Mode	Color	
0	srcCopy	8	patCopy	33	Black
1	srcOr	9	pator	30	White
2	srcXor	10	patXor	205	Red
3	srcBic	11	PatBic	341	Green
4	notsrcCopy	12	notpatCopy	409	Blue
5	notsrcOr	13	notpatOr	273	Cyan
6	notsrcXor	14	notpatXor	137	Magenta
7	notsrcBic	15	notpatBic	69	Yellow

Also see PEN in this appendix.

GLOBAL MEMORY LOCATIONS USEFUL WITH QUICKDRAW

Offsets off Application A5 Globals: A5=PEEK LONG(&904)

PEEK	Size	Name	Description	
A5-4	Long	GrafPort	Pointer to current GrafPort	
A5-12	Byte8	White	White Pattern	(8 bytes)
A5-20	byte8	Black	Black Pattern	(8 bytes)
A5-28	Byte8	Gray	50% Gray Pattern	(8 bytes)
A5-36	Byte8	ltGray	25% Gray Pattern	(8 Bytes)
A5-44	Byte8	dkGray	75% Gray Pattern	(8 Bytes)
A5-112	Byte68	Cursor	Current Mouse Cursor	(68 Bytes)
A5-126	Byte14	BitMap	Full Screen BitMap	(14 Bytes)
A5-128	Long	RandSeed	Random Seed for Mac's Random #	
A5-126	Long	BaseAddr	Base Address of Screen Bit Image	
A5-122	Word	rowBytes	Row Width in Byte (Even # of Bytes)	
A5-120	Rect	Bounds	Screen Bounds Rectangle	

Example: To obtain RandSeed above use: PEEK LONG(PEEK LONG(&904) - 128).

MACINTOSH APPENDIX

GrafPort POINTER

The GrafPort Pointer points to many important areas of information for a specific window. Each window has it's own GrafPort to keep track of cursor and pen position, current color etc.

To get the GrafPort pointer to a window use the ZBasic GET WINDOW or the toolbox GETPORT.

Examples:

TO GET — **USE THIS PEEK MODE**

pnMode

```
GET WINDOW #1, GrfPtr&:  REM This gets the pointer
PEEK WORD(GrfPtr&+56)
```

PenSize

```
PEEK LONG(GrfPtr&+52)
```

PenPat

```
FOR X=0 TO 7
  PRINT BIN$( PEEK(GrfPtr&+58+X))
NEXT
REM  BIN$ shows the bit pattern in 0's and 1's
```

---------- GrfPtr: GrafPort Ptr Offsets---------- 206 Bytes ----------

Offset	Type	Name	Description
0	Word	Device	Device-specific information
2	Long	baseAddr	Base Address to bit image
6	Word	RowBytes	Width of full image in bytes (even number)
8	Rect	PortRect	Boundary Rectangle of image
16	Rect	PortRect	GrafPort Rectangle
24	Long	VisRgn	Visable Region
28	Long	ClipRgn	Clipping Region
32	Byte 8	bkPat	Background pattern (pattern is 8 bytes)
40	Byte 8	fillPat	Fill pattern (pattern is 8 bytes)
48	Point	PenLoc	Pen Location
52	Long	PenSize	Pen Size (first Word is Y, second Word is X)
56	Word	pnMode	Pen Mode (see above)
58	Byte 8	PenPat	Pen Pattern (see PEN above and in appendix)
66	Word	pnVis	Pen Visability
68	Word	txFont	Font Number
70	Word	txFace	Font Face
72	Word	txMode	Font Transfer Mode
74	Word	txSize	Font Size (1 to 127)
76	Fixp	spExtra	Space extra fixed Point (+/-32767.xxxx)
80	Long	fgColor	Foreground color
84	Long	bkColor	Background color
88	Word	ColrBit	Color bit (used by printer manager)
90	Word	PatStretch	Used Internally*
92	Long	PicSave	Handle to PICTURE being saved
96	Long	rgnSave	Handle to Region being saved
100	Long	PolySave	Handle to Polygon being saved
104	....	GrafProcs	

*See "Inside Macintosh" for definitions to the other offsets which are normally not used (not enough room in this manual to describe them in the detail necessary).

MACINTOSH APPENDIX

QuickDraw FUNCTIONS AND PROCEDURES GROUPED BY TYPE

GrafPort

CALL	INITGRAF (globe)	Don't Use ***
CALL	OPENPORT (GrfPtr)	Allocates space for GrafPort***
CALL	INITPORT (GrfPtr)	Initializes GrafPort Information***
CALL	CLOSEPORT (GrfPtr)	Closes GrafPort & Releases Memory***
CALL	GETPORT (GrfPtr&)	Get current Grafport pointer and return in Grfptr& (returns GrafPort to current output Window)
CALL	GRAFDEVICE (Word)	Sets the Device field in GrafPort to Word***
CALL	SETPORTBITS (BitMap)	Change portbits field of GrafPort***
CALL	PORTSIZE (xSize, ySize)	Change current port size (active area)
CALL	MOVEPORTTO (xPos, yPos)	Move Top/Left corner of current GrafPort
CALL	SETORIGIN (xPos, yPos)	Set a New Graphic origin (normally 0,0)
CALL	SETCLIP (RgnHnd)	Set Clipping Region of GrafPort to RgnHnd
CALL	GETCLIP (RgnHnd)	Get Current GrafPort clip Region to RgnHnd
CALL	CLIPRECT (Rect)	Changes the GrafPort Clip Region to Rect See example under PICTURE in this appendix.

***May conflict with ZBasic: Use with caution.

COLOR

CALL	FORECOLOR (Long)	Set Foreground color to Long value
CALL	BACKCOLOR (Long)	Set Background color to Long value
CALL	COLORBIT (Word)	Set Color Bit (used by ImageWriter™ printer driver)

Note: ZBasic's COLOR statement does this automatically for printing to the ImageWriter™.

PEN

CALL	PENNORMAL	Reset Pen to 1,1 pixels,patCopy,Black
CALL	PENPAT (Pat)	Set Current Pen Pattern (Pat= 8 bytes)
CALL	BACKPAT (Pat)	Changes the background pattern to Pat
PHndl=	FN GETPATTERN (ID)	Get Handle to Pattern Resource # ID
CALL	HIDEPEN	Stop drawing Until matching showpen
CALL	SHOWPEN	Resume Drawing in current GrafPort

Note: These two procedures are stacked (you must do one SHOWPEN for every HIDEPEN).

CALL	GETPEN (VPoint)	Returns current Pen Location in Point
CALL	GETPENSTATE (PenRec)	Get Current Pen State into PenRec
CALL	SETPENSTATE (PenRec)	Set Current Pen State With PenRec
CALL	PENSIZE (xSize, Ysize)	Set Pen Size in Pixels
CALL	PENMODE (Pmode)	Set Pattern Transfer Mode (8 to 15)

continued...

MACINTOSH APPENDIX

continued from previous page...

EXAMPLE

```
REM Example of SHOWPEN, HIDEPEN and PENSIZE
REM SX = showpen x coordinate.
REM SY = showpen y coordinate.
REM HX = hidepen x coordinate.
REM HY = hidepen y coordinate.
CLS: SX=10: SY=20: HX=SX: HY=SY: X=SX: Y=100
FOR Z=1TO20
  CALL PENSIZE (Z,Z):REM INCREMENTS PEN WIDTH AND HEIGHT
  PLOT X,Y
  LONG IF (Z MOD 2)=0
    CALL HIDEPEN: PLOT HX,HY: CALL SHOWPEN
  XELSE
    PLOT SX,SY
  END IF
  SX=SX+50:X=SX
NEXT
```

MOVE AND LINE

CALL	MOVE(dx,dy)	Move relative to Current Position. If dx=10, dy=20 and current position is 20,30 then moves to 30,50.
CALL	MOVETO (xPos, yPos)	Move Pen Pointer to xPos,Ypos
CALL	LINE (dx, dy)	Draw Line relative to x+dx, y+dy
CALL	LINETO (xPos, yPos)	Draw Line from current Pos to xPos,Ypos

EXAMPLE

```
WINDOW#1,"Example of MOVETO and LINETO"
COORDINATE WINDOW: REM All QuickDraw routines use "Pixels"
CALL MOVETO (0,0)
DO
  CALL LINETO (RND(WINDOW(2)), RND(WINDOW(3)))
UNTIL LEN(INKEY$)
```

RECTANGLES

CALL	FRAMERECT(Rect)	OutLine Rectangle
CALL	PAINTRECT(Rect)	Draw Rectangle in Foreground
CALL	ERASERECT(Rect)	Erase Rectangle to Background
CALL	INVERTRECT(Rect)	Invert Every bit in the Rectangle
CALL	FILLRECT(Rect,Pat)	Fill Rect with Pattern Specified
CALL	OFFSETRECT (Rect, dx, dv)	Offset rectangle by dx,dy (Same size)
CALL	INSETRECT (Rect, dx, dy)	Shrink or Expand Rectangle by dx,dy
CALL	UNIONRECT (SRect1, SRect2, Rect)	Returns the Union of SRect1,2 in Rect
CALL	PT2RECT (Point, Point, Rect)	Returns the Smallest Rect from 2 points
CALL	SETRECT (VRect,Top, Left,Bottom,Right)	Set Rectangle Values into VRect
Byte=	FN SECTRECT (SRect1, SRect2, Rect)	If Byte=True, returns intersection in Rect
Byte=	FN PTINRECT (Point, Rect)	Returns True if Point in Rectangle
Byte=	FN EQUALRECT (Rect1, Rect2)	Returns True if Rect1=Rect2
Byte=	FN EMPTYRECT (Rect)	Returns True if Rect too small or invisable

continued...

MACINTOSH APPENDIX

ROUND CORNERED RECTANGLES

Other Terms used in this section:

Oh	Word	Height of rounded edge in Pixels
Ow	Word	Width of rounded edge in Pixels

CALL	FRAMEROUNDRECT(Rect,Ow,Oh)	OutLine Roundrect
CALL	PAINTROUNDRECT(Rect,Ow,Oh)	Draw Roundrect in ForeGround
CALL	ERASEROUNDRECT(Rect,Ow,Oh)	Erase Roundrect in Background
CALL	INVERTROUNDRECT(Rect,Ow,Oh)	Invert Every bit in Roundrect
CALL	FILLROUNDRECT(Rect,Ow,Oh,Pat)	Fill Roundrect with Pattern Specified

EXAMPLE1

```
CLS: DIM T,L,B,R,PAT%(3): REM All QuickDraw routines use "Pixels"
PAT%(0)=&AA55 : PAT%(1)=&AA55 : PAT%(2)=&AA55 : PAT%(3)=&AA55
T=100 : L=100 : B=250 : R=300
CALL FRAMERECT(T)                       :GOSUB "WAIT/CLS"
CALL PAINTRECT(T)                       :GOSUB "WAIT/CLS"
CALL INVERTRECT(T)                      :GOSUB "WAIT/CLS"
CALL FILLRECT(T,PAT%(0))                :GOSUB "WAIT/CLS"
CALL FRAMEOVAL(T)                       :GOSUB "WAIT/CLS"
CALL PAINTOVAL(T)                       :GOSUB "WAIT/CLS"
CALL INVERTOVAL(T)                      :GOSUB "WAIT/CLS"
CALL FILLOVAL(T,PAT%(0))                :GOSUB "WAIT/CLS"
CALL FRAMEROUNDRECT(T,10,10)            :GOSUB "WAIT/CLS"
CALL PAINTROUNDRECT(T,20,20)            :GOSUB "WAIT/CLS"
CALL INVERTROUNDRECT(T,30,30)           :GOSUB "WAIT/CLS"
CALL FILLROUNDRECT(T,40,40,PAT%(0))     :GOSUB "WAIT/CLS"
CALL FRAMEARC(T,0,45)                   :GOSUB "WAIT/CLS"
CALL PAINTARC(T,45,90)                  :GOSUB "WAIT/CLS"
CALL INVERTARC(T,90,135)                :GOSUB "WAIT/CLS"
CALL FILLARC(T,135,180,PAT%(0))         :GOSUB "WAIT/CLS"
END
"WAIT/CLS" : DELAY 100 : CLS : DELAY 100 : RETURN
```

EXAMPLE2

```
REM T%, L% = Top left corner
REM B%, R% = Bottom right corner
REM OX = offset X pixels horizontally.
REM OY = offset Y pixels vertically.
TRONB: REM All QuickDraw routines use "Pixels"
COORDINATE WINDOW:CLS
DIM T%,L%,B%,R%
T%=0:L%=0:B%=120:R%=90:OX=15
FOR I=0TO37
  PEN,,,,I
  :
  REM   Change PAINT on next line to ERASE, FRAME, INVERT or FILL
  REM   to see other ways of doing rectangles. You can also
  REM   replace RECT with ROUNDRECT to do rounded corners.
  :
  CALL PAINTRECT (T%):  DELAY 100
  LONG IF I>15
    LONG IF OX<=0
      OX=-15: OY=0
    END IF
    OY=OY+1: OX=OX-1
  END IF
  CALL OFFSETRECT (T%,OX,OY)
NEXT:END
```

MACINTOSH APPENDIX

OVALS

CALL	FRAMEOVAL(Rect)	OutLine Oval
CALL	PAINTOVAL(Rect)	Draw Oval in ForeGround
CALL	ERASEOVAL(Rect)	Erase Oval in Background
CALL	INVERTOVAL(Rect)	Invert Every bit in Oval
CALL	FILLOVAL(Rect,Pat)	Fill Oval with Pattern Specified

EXAMPLE

```
TRONB: COORDINATE WINDOW:CLS
DIM T%,L%,B%,R%:    REM This sets up the Rect coordinates
T%=10:L%=10:B%=200:R%=300
:
FOR I=0TO37
  PEN,,,,I
  :
  REM   Change PAINT on next line to ERASE, FRAME, INVERT OR FILL
  REM   to see other ways of doing rectangles.
  :
  CALL PAINTOVAL (T%)
NEXT
END
```

ARCS

Other Terms used in this section:

Strt	Word	Start Angle 0 to 360	(0=top)
Angle	Word	Number of Degrees ClockWise	(0 to 360)

CALL	FRAMEARC(Rect,Strt,Angle)	OutLine Arc (no lines to center)
CALL	PAINTARC(Rect,Strt,Angle)	Draw Arc in ForeGround
CALL	ERASEARC(Rect,Strt,Angle)	Erase Arc in Background
CALL	INVERTARC(Rect,Strt,Angle)	Invert every bit in Arc
CALL	FILLARC(Rect,Strt,Angle,Pat)	Fill Arc with Pattern Specified

EXAMPLE

```
COORDINATE WINDOW: CLS:REM All QuickDraw routines use "Pixels"
DIM T%,L%,B%,R%
T%=50:L%=150:B%=250:R%=250:      REM Set up Rect
FOR I=0 TO 37 STEP 4
  PEN,,,,I
  CALL PAINTARC (T%,0,90)
  PEN,,,,I+1
  CALL PAINTARC (T%,90,90)
  PEN,,,,I+2
  CALL PAINTARC (T%,180,90)
  PEN,,,,I+3
  CALL PAINTARC (T%,270,90)
NEXT
```

MACINTOSH APPENDIX 

REGIONS

Regions are limited to 32K in length. Do not call OPENRGN when another region is already open as all previously defined regions will then act erradically.

Other Terms used in this section:

SRgn	RgnHnd	Source Region Handle
DRgn	RgnHnd	Destination Region Handle

RgnHnd=FN NEWRGN		Create a New Empty Region Handle
Byte=	FN PTINRGN (Point, RgnHnd)	True if Point inside Region
Byte=	FN RECTINRGN (Rect, RgnHnd)	True if all or part of Rect in Region
Byte=	FN EQUALRGN (RgnHnd1, RgnHnd2)	True if Region #1 = Region #2
Byte=	FN EMPTYRGN (RgnHnd)	True if Region is empty
CALL	DISPOSERGN (RgnHnd)	Releases Memory Ocupied by RgnHnd
CALL	COPYRGN (SRgn, DRgn)	Copys Region Structure SRgn to Drgn
CALL	SETEMPTYRGN (RgnHnd)	Makes RgnHnd Point to Empty Region
CALL	SETRECTRGN(RgnHnd,Top,Bottom,Left,Right)	Set Region to Rectangle specified
CALL	RECTRGN (RgnHnd, Rect)	Sets Region to Rectangle only
CALL	OPENRGN	Tell QuickDraw to Save Region Information
CALL	CLOSERGN (RgnHnd)	Closes Open Region and save in RgnHnd
CALL	OFFSETRGN (RgnHnd, dx, dy)	Offsets Region by dx,dy
CALL	INSETRGN (RgnHnd, dx, dy)	Shrinks or Expands Region, +n=inward
CALL	SECTRGN (SRgn, SRgn, DRgn)	Calculates Intercetion of 2 Regions
CALL	UNIONRGN (SRgn, SRgn, DRgn)	Calculates Union of 2 Regions
CALL	DIFFRGN (SRgn1, SRgn2, DRgn)	Calculates Difference of 2 Regions
CALL	XORRGN (SRgn1, SRgn2, DRgn)	Calculates XORof 2 Regions
CALL	FRAMERGN (RgnHnd)	Outline Area Specified by RgnHnd
CALL	PAINTRGN (RgnHnd)	Paint Area Specified by RgnHnd
CALL	ERASERGN (RgnHnd)	Erase Area Specified by RgnHnd
CALL	INVERTRGN (RgnHnd)	Invert Area Specified by RgnHnd
CALL	FILLRGN (RgnHnd,Pat)	Fill Area Specified by RgnHnd

EXAMPLE

```
WINDOW#1,"Example of using REGIONS"
COORDINATE WINDOW: REM All QuickDraw routines use "Pixels"
DIM PAT%(3), T%, L%, B%, R%:REM Set up variables for Rect
PAT%(0)=&AA55 : PAT%(1)=&AA55 : PAT%(2)=&AA55 : PAT%(3)=&AA55
:
Barbell&=FN NEWRGN
CALL OPENRGN
  CALL SETRECT(T%, 20,20,30,50)      :REM Form the weight
  CALL FRAMEOVAL (T%)
  CALL SETRECT(T%, 25,30,85,40)      :REM Form the BAR
  CALL FRAMERECT (T%)
  CALL SETRECT(T%, 80,20,90,50)      :REM Form the right-weight
  CALL FRAMEOVAL (T%)
CALL CLOSERGN(Barbell&)
CALL FILLRGN(Barbell&, PAT%(0))
CALL OFFSETRGN(Barbell&, 100,100)    :REM Offset and do again
PAT%(0)=&FFFF : PAT%(1)=&FFFF : PAT%(2)=&FFFF : PAT%(3)=&FFFF
CALL FILLRGN(Barbell&, PAT%(0))
CALL OFFSETRGN(Barbell&, 100,100)    :REM Offset and do again
CALL FRAMERGN(Barbell&)              :REM Do an outline
CALL DISPOSERGN(Barbell&)            :REM Kill Region
DO: UNTIL LEN(INKEY$): END
```

MACINTOSH APPENDIX

SCROLL AND COPYBITS

```
CALL   SCROLLRECT(Rect,dx,dy,URgn)                    Move Pixels dx,dy Update Region=URgn
CALL   COPYBITS(SRect,DRect,SBit,DBit,Mode,MaskRgn)   Copy Source Rect area to the Dest. Rect
```

PICTURE (Also see PICTURE in this appendix)

Other Terms used in this section:

PicHnd	Long	Handle to Picture	(PICTURES are limited to 32K)
Kind	Word	Type of Comment	(Comment Number)
Size	Word	# of Bytes	(0 if No Data)
Data	Long	Handle to actual Data	(Nil if No Data)

---------- PICTURE Record ----------

Offset	Size	Name	Description
0	Word	picSize	Total Size in Bytes (up to 32767)
2	Rect	picFrame	Bounding Rectangle
10	Data	picData	Picture Byte Codes (Data Making up Picture)

```
PicHnd= FN OPENPICTURE(Rect)          Open Picture size Rect to store Graphics
PicHnd= FN GETPICTURE(ID)             Get Handle to Resource Picture With ID
CALL    CLOSEPICTURE                  Tell QuickDraw to stop saving Picture
CALL    PICCOMMENT(Kind,Size,Data)    Save Comment into Picture Structure
CALL    DRAWPICTURE(PicHnd,Rect)      Draw Picture scaled to Fit Rect
CALL    KILLPICTURE(PicHnd)           Release Memory Occupied by Picture
```

MOUSE CURSOR

```
CHndl=  FN GETCURSOR(ID)              Get a Handle to Resource Cursor ID
CALL    INITCURSOR                    Change Mouse Cursor Back to Arrow
CALL    SHIELDCURSOR(Rect,Point)      Hide Cursor if it Intercects Rect
CALL    SETCURSOR(Curs)               Change Mouse Cursor to Curs Pattern
CALL    HIDECURSOR                    Hide Mouse Cursor
CALL    SHOWCURSOR                    Show Mouse Cursor
CALL    OBSCURECURSOR                 Hide Mouse Cursor until Moved
```

See example under CURSOR in the reference section of this appendix.

MACINTOSH APPENDIX 

POLYGONS

Other Terms used in this section:

PlyHnd	Long	Handle to Poly

PlyHnd=	FN OPENPOLY	Returns Handle to Poly & starts saving
CALL	CLOSEPOLY	Tell QuickDraw to Stop Saving Poly
CALL	KILLPOLY(PlyHnd)	Release Memory Occupied by Poly
CALL	OFFSETPOLY(PlyHnd,dx,dy)	Offset Poly Definition by dx,dy
CALL	MAPPOLY(PlyHnd,Var,Var)	Map Area specified by Poly Handle
CALL	FRAMEPOLY(PlyHnd)	Outline Area specified by Poly Handle
CALL	PAINTPOLY(PlyHnd)	Paint Area specified by Poly Handle
CALL	ERASEPOLY(PlyHnd)	Erase Area specified by Poly Handle
CALL	INVERTPOLY(PlyHnd)	Invert Area specified by Poly Handle
CALL	FILLPOLY(PlyHnd,Pat)	Fill Area specified by PlyHnd with Pattern

EXAMPLE

```
TRONB: TEXT,,,0
COORDINATE WINDOW: CLS : DIM A%(10)
X&=FN OPENPOLY: CALL SHOWPEN
PLOT 0,0 TO 200,0: DELAY 800
PLOT 200,0 TO 200,200: DELAY 800
PLOT 200,200 TO 100,100: DELAY 800
PLOT 100,100 TO 100,200: DELAY 800
PLOT 100,200 TO 0,0: DELAY 800
CALL CLOSEPOLY
FOR I=0 TO 37
  PEN,,,,I
  CALL PAINTPOLY(X&):PRINT@(50,5)"PAINT POLY..."
  DELAY 100
  CALL INVERTPOLY(X&):PRINT@(50,5)"INVERT POLY..."
  DELAY 100
NEXT
CALL KILLPOLY(X&)
```

GRAPHIC UTILITIES

Other Terms used in this section:

VPoint	Long	Variable With Point Value in it
VRect	Rect	Variable Rect Modified by ToolBox Call
h	Word	Horizontal Position
v	Word	Vertical Position

Byte=	FN EQUALPT (Point, Point)	True if Point #1 = Point #2
Byte=	FN GETPIXEL (xPos, YPos)	Get pixel state: NOT zero if pixel is set.
CALL	ADDPT(Point,VPoint)	Add Point to VPoint stored in VPoint
CALL	SUBPT(Point,VPoint)	Sub Point to VPoint stored in VPoint
CALL	SETPT(VPoint,h,v)	Make Point from h,v as Words
CALL	LOCALTOGLOBAL(VPoint)	Convert Local in VPoint to Global Point
CALL	GLOBALTOLOCAL(VPoint)	Convert Global in VPoint to Local Point
CALL	STUFFHEX(Ptr,Str)	Convert Hex in Str to Bytes at Ptr
CALL	SCALEPT(VPoint,SRect,DRect)	Scales Point Size in VPoint
CALL	MAPPT(VPoint,SRect,DRect)	Map/Scale Point from Source to Dest Rect
CALL	MAPRECT(VRect,SRect,DRect)	Map/Scale VRect from Source to Dest Rect
CALL	MAPRGN(RgnHnd,Var,Var)	Map/Scale Region from Source to Dest Rect

MACINTOSH APPENDIX

ICONS

IHndl=	FN GETICON(ID)	Get Handle to resource ICON in IHndl
CALL	PLOTICON(Rect,IHndl)	Draw ICON Sized and at Rect

EXAMPLE

```
REM This example shows how to to access
REM ICONS from our application and display them
REM in your programs. Courtesy A.G.
:
CLS : DIM T,L,B,R
"LOOP"
FOR J=0 TO 2
  FOR I=1 TO 128 STEP 4
    T=140-I : L=256-I : B=140+I : R=256+I
    CALL PLOTICON(T,FN GETICON(J))
  NEXT
NEXT
TRONX : GOTO "LOOP"
```

QUICKDRAW ROUTINES AVAILABLE ON NEW ROMS ONLY

Other Terms used in this section:

SPtr	Long	Source Pointer
DPtr	Long	Destination Pointer
SRow	Word	Source Row
DRow	Word	Destination Row
Hght	Word	Hight (in Pixels)
Wdth	Word	Width (in Words)
Seedh	Word	Vertical Offset in Pixels
SeedV	Word	Horizontal Offset in Pixels
SBit	Long	Source BitMap
MBit	Long	Mask BitMap
DBit	Long	Destination BitMap
MRect	Var	Mask Rectangle

CALL SEEDFILL(SPtr,DPtr,SRow,DRow,Hght,Wdth,Seedh,Seedv)
CALL CALCMASK(SPtr,DPtr,SRow,DRow,Hght,Wdth)
CALL COPYMASK(SBit,MBit,DBit,SRect,MRect,DRect)
Long= FN GETMASKTABLE

MACINTOSH APPENDIX

TEXT AND FONTS

See TEXT statement in this appendix for more information about using Macintosh fonts.

Terms used in this section:

Fnum	Word	Text Font Number
Fsize	Word	Font Size 1 to 127
Face	Word	Text Face bits (bold, italic etc.)
Fmod	Word	Text Mode (Mode 0 to 7) See bug note under TEXT in this appendix.
Fixp	Long	Fixed Point Example: F&=num#*65536
Pxls	Word	Number of Pixels

FntRec	Var	Font Record 8 Bytes		--- 128k Rom FFrec ---
0	Ascent	0	Ascent Fixed	
2	Descent	4	Decent Fixed	
4	WidMax	8	Leading Fixed	
6	Leading	12	Width Max Fixed	
16	Hndl Width Table			

Bit ---	Value -	Face ---	Num ---	Font -----	Num ---	Font ----
0	1	Bold	0	System	1	Appl Font
1	2	Italic	2	New York	3	Geneva
2	4	Underline	4	Monaco	5	Venice
3	8	OutLine	6	London	7	Athens
4	16	Shadow	8	SanFransisco	9	Toronto
5	32	Condensed	11	Cairo	12	LosAngeles
6	64	Extended	20	Times	21	Helvetica
7	128	*Reserved*	22	Courier	23	Symbol

Pxls%=	FN CHARWIDTH(Char)	Returns Pixel Width of Char (ASCII Val)
Pxls%=	FN STRINGWIDTH(Str)	Returns Pixel Width of Str
Pxls%=	FN TEXTWIDTH(TxtPtr,Spos,Cnt)	Returns Pixel Width of text,Spos,cnt
Byte=	FN REALFONT(Fnum,Fsize)	Givin Font Number/Size True if in Sys
Long=	FN SWAPFONT(Var)	*** Don't Use ***
CALL	TEXTFONT(fnum)	Change current font to Font Number word
CALL	TEXTFACE(Face)	Change Text Face to face
CALL	TEXTMODE(Fmod)	Change Text Mode (Fmod= 0 to 7)
CALL	TEXTSIZE(Fsize)	Change text Size (Fsiz= 1 to 127)
CALL	SPACEEXTRA(Fixp)	Change Character Spacing +/-nnn.nnn
CALL	DRAWCHAR(Char)	Draw Ascii Char at current pen position
CALL	DRAWSTRING(Str)	Draw String from current pen Position
CALL	DRAWTEXT(TxtPtr,Spos,Cnt)	Draw Text from current pen Position
CALL	GETFONTINFO(FntRec)	Get Font Information into font record
CALL	INITFONTS *** Don't Use ***	
CALL	GETFONTNAME(Fnum,Name$)	Returns Name for Font Number in Name$
CALL	GETFNUM(Name,Fnum%)	Returns Font Number for Font Name in Fnum%
CALL	SETFONTLOCK(Byte)	If Byte <> 0 Current Font Un-Purgable

TEXT for128k ROM systems only

CALL	MEASURETEXT(Cnt,Str,Var%)	Makes an Integer Array of Pixel Positions
CALL	SETFSCALEDISABLE(Word)	True Tells QuickDraw Not to scale fonts
CALL	FONTMETRICS(FFrec)	Fixed Point Version of GetFontInfo

MACINTOSH APPENDIX

WINDOW MANAGER

Terms used in this section:

WPtr	Long	Window Pointer ZBasic™: Wptr&=WINDOW(14)
RefCon	Long	Window Long word Ref Value
title	Str	Window Title
wID	Word	ID of resource Window
Wsto	Long	Area to Store Window record (Nil in Heap)
BWptr	Long	Behind Which Window 'Wptr'
Vis	Byte	True if Visable
pID	Word	Proc ID, Window Definition
gWay	Byte	True if GoAway Box
dxy	Long	Delta x,y (0 if no change)
Where	Word	Where the point is on the screen (Global Coordinates).

Where	Point in	pID	Window Proc ID's	Offset	Description
0	Desk Top	0	Std Document	0	GrafPort
1	Menu Bar	1	Alert or Dialog	208	WindowKind
2	System Window	2	Plain Box	210	Visable
3	Contents	3	Plain W/Shadow	211	HiLited
4	Drag	4	Doc Wo/Size Box	212	GoAway
5	Grow Box	8	Zoom Window	213	*Reserved*
6	GoAway	16-23	Rounded Edge DocProc	214	StructureRgn
7	Zoom-In Box				
8	Zoom-Out Box				

WPtr=	FN NEWWINDOW(wSto,Rect,Title,Vis,pID,BWptr,gWay,RefCon)	
WPtr=	FN GETNEWWINDOW(wID,wSto,BWptr)	Opens up Resource Window Definition
WPtr=	FN FRONTWINDOW	Returns Window pointer of Front Window
Where=	FN FINDWINDOW(Point,Wptr&)	Returns Point in Window. See 'Where'
Byte=	FN TRACKGOAWAY(Wptr,Point)	Track Mouse Down in GoAway Box
dxy=	FN GROWWINDOW(Wptr,Point,Rect)	Track Window Grow,Mouse Down in Grow
RefCon=	FN GETWREFCON(Wptr)	Get Current RefCon Value
PicHnd=	FN GETWINDOWPIC(Wptr)	Get Handle to Window Picture
Point=	FN PINRECT(Rect,Point)	Pins the Point Inside the Rect
Long=	FN DRAGGRAYRGN(Long,Point,Var,Var,Word,Long)	
CALL	INITWINDOWS	*** Don't Use! ***
CALL	GETWMGRPORT(Wptr&)	Returns Pointer to Window Mgr Window
CALL	CLOSEWINDOW(Wptr)	Close Window Pointed to by Wptr
CALL	DISPOSEWINDOW(Wptr)	Closes Window Created With Nil as wStorage
CALL	SETWTITLE(Wptr,Str)	Change Window Title to Str
CALL	GETWTITLE(Long,Var$)	Get Current Window title in Var$
CALL	SELECTWINDOW(Wptr)	Makes this Window the Front Window
CALL	HIDEWINDOW(Wptr)	Makes the Window invisable
CALL	SHOWWINDOW(Wptr)	Makes the Window Visable (Same Order)
CALL	SHOWHIDE(Wptr,Byte)	If Byte=0 Make Invisable else Visable
CALL	HILITEWINDOW(Wptr,Byte)	if Byte=0 UnHilite else Hilite Window
CALL	BRINGTOFRONT(Wptr)	Brings the Window in front of Others
CALL	SENDBEHIND(Wptr1,Wptr2)	Sends Window1 behind Window2
CALL	DRAWGROWICON(Wptr)	Draws Window Grow icon
CALL	MOVEWINDOW(Wptr,gh,gv,Byte)	Move Window to Global gh,gv,Front if true
CALL	DRAGWINDOW(Wptr,Point,Rect)	Drag Window,Mouse Point,Bounds Rectangle

continued...

MACINTOSH APPENDIX

continued from previous page...

CALL	SIZEWINDOW(Wptr,w,h,Byte)	Size to w,h, UpDate Event if Byte True
CALL	INVALRECT(Rect)	Force an UpDate Event For Rect Area
CALL	INVALRGN(RgnHnd)	Force an UpDate Event for Rgn Area
CALL	VALIDRECT(Rect)	Tell Window Manager this Rect Valid
CALL	VALIDRGN(RgnHnd)	Tell Window Manager this Rgn Valid
CALL	BEGINUPDATE(Wptr)	Begin Update Process for Window
CALL	ENDUPDATE(Wptr)	End Update Process for Window
CALL	SETWREFCON(Wptr,RefCon)	Set Window RefCon Field
CALL	SETWINDOWPIC(Wptr,PicHnd)	Set Window Picture Handle
CALL	CLIPABOVE(Wptr)	Sets up Clip Region
CALL	SAVEOLD(Wptr)	Save Content/Structure area of Window
CALL	DRAWNEW(Wptr,Byte)	Restore from Above, Update if True
CALL	PAINTONE(Wptr,RgnHnd)	See Inside Mac
CALL	PAINTBEHIND(Wptr,RgnHnd)	See Inside Mac
CALL	CALCVIS(Wptr)	See Inside Mac
CALL	CALCVISBEHIND(Wptr,RgnHnd)	See Inside Mac
CALL	SETPORT(Wptr)	See Inside Mac
*****	**128k Roms Only**	*****
CALL	ZOOMWINDOW(Wptr,Point,Word)	See Inside Mac
Byte=	FN TRACKBOX(Wptr,Word,Byte)	See Inside Mac

EXAMPLE

```
REM This example shows you where the mouse
REM is on the screen:
:
DIM Y%,X%:COORDINATE WINDOW
WINDOW#1,"Window Pointer Example"
GET WINDOW#1, Wptr&
REM Use CALL GETPORT (Wptr&) to get current window GrfPtr&.
DIALOG ON:BREAK ON
DO
  CALL GETMOUSE(Y%)
  CALL LOCALTOGLOBAL(Y%)
  WHERE=FN FINDWINDOW(Y%,Wptr&)
  PRINT X%,Y%,
  SELECT CASE WHERE
    CASE 0:PRINT"DESK": CURSOR 2
    CASE 1:PRINT"MENU BAR"
    CASE 2:PRINT"SYSTEM WINDOW (Desk Accessory?)"
    CASE 3:PRINT"CONTENTS":CURSOR 0
    CASE 4:PRINT"DRAG":CURSOR 4
    CASE 5:PRINT"GROW BOX"
    CASE 6:PRINT"GO-AWAY BOX":CURSOR 3
    CASE 7:PRINT"ZOOM-IN BOX"
    CASE 8:PRINT"ZOOM-OUT BOX"
    CASE ELSE:PRINT"NUMBER RETURNED WAS:"WHERE"--->UNKNOWN!"
  END SELECT
UNTIL LEN(INKEY$)
END
```

MACINTOSH APPENDIX

CONTROL MANAGER

This section covers things like Buttons, Scroll bars and other controls.

Terms used in this section:

Wprt	Long	Window Pointer
title	Str	String for Control Title
Value	Word	Current control Value
Min	Word	Minumum control Value
Max	Word	Maximum control Value
RefCon	Long	Controls Ref Value
ProcID	Word	Control Definition Procedure ID
Visable	Byte	True if Visable
cID	Word	Contol Resource ID number
Chndl	Long	Control Handle
Aproc	Long	Pointer to Action Procedure
Axis	Word	1=Horz Move,2=Vertical Move,0=Both
Slopr	Rect	Slop Rectangle
Lrect	Rect	Limit Rectangle
Horz	Word	Size in Pixels Width
Vert	Word	Size in Pixels Height

CONTROL HANDLE

---------- Chndl: Control Handle Template ----------

Offset	Size	Name	Description
0	hndl	nextControl	next control in the list
4	WPtr	contrlOwner	owning window
8	Rect	contrlRect	bounding rectangle
16	Byte	contrlVis	visible state
17	Byte	contrlHilite	Highlight state
18	Word	contrlValue	current value of control
20	Word	contrlMin	minimum value of control
22	Word	contrlMax	maximum value of control
24	Hndl	contrlDefHandle	control definition procedure
28	Hndl	contrlData	data for definition proc
32	Proc	contrlAction	local actionProc
36	Long	contrlRFcon	refcon defined by application
40	str	contrlTitle	title string

Control Manager Definitions

ID	Type	Font Used	Part Code	Description
0	Button	System	10	Simple Button
1	Check Box	System	11	Check Box or Radio
2	Radio Button	System	20	Up Arrow Scroll Bar
8	Button	Window	21	Down Arrow
9	Check Box	Window	22	Page up Area
10	Radio Button	Window	23	Page Down Area
16	Scroll Bar	none.	129	In Scroll Thumb

Chndl=	FN NEWCONTROL(Wptr,Rect,title,Visable,Value,Min,Max,ProcID,RefCon)	
Chndl=	FN GETNEWCONTROL(cID,Wptr)	Get and open a resource defined Control
CALL	DISPOSECONTROL(Chndl)	Releases Memory and Deletes Control
CALL	KILLCONTROLS(Wptr)	Dispose of all Controls in Window
Word=	FN TESTCONTROL(Chndl,Point)	Returns Part code of where Point is

continued...

MACINTOSH APPENDIX

continued from previous page...

Word=	FN FINDCONTROL(Point,Wptr,Chndl&)	Get Chndl with Point in Window
Word=	FN TRACKCONTROL(Chndl,Point,Aproc)	Use when Mouse down in Control
CALL	HIDECONTROL(Chndl)	Makes Control Invisable
CALL	SHOWCONTROL(Chndl)	Makes Control Visable
CALL	DRAWCONTROLS(Wptr)	Draws all Controls in Window
CALL	HILITECONTROL(Chndl,Word)	Changes Control HiLite 255=Inactive...
CALL	MOVECONTROL(Chndl,xPos,yPos)	Move Top/Left corner to new Window Loc
CALL	DRAGCONTROL(Chndl,Point,Lrect,Slopr,Axis)	User Drag Control Proc
CALL	SIZECONTROL(Chndl,horz,vert)	Size Control to Horz width,Vert Height
Value=	FN GETCTLVALUE(Chndl)	Get Current Control Value
Min=	FN GETCTLMIN(Chndl)	Get Current Control Min
Max=	FN GETCTLMAX(Chndl)	Get Current Control Max
RefCon=	FN GETCREFCON(Chndl)	Get Current Control RefCon
AProc=	FN GETCTLACTION(Chndl)	Get Current Control Action Proc Pointer
CALL	GETCTITLE(Chndl,Var$)	Get Current Control title into Var$
CALL	SETCTLVALUE(Chndl,Value)	Set Control Value
CALL	SETCTLMIN(Chndl,Min)	Set Control Minimum
CALL	SETCTLMAX(Chndl,Max)	Set Control Maximum
CALL	SETCREFCON(Chndl,Refcon)	Set Control RefCon
CALL	SETCTLACTION(Chndl,Aproc)	Set Control Action Procedure
CALL	SETCTITLE(Chndl,title)	Set Control title
*****	**128k ROM Only**	*****
CALL	UPDTCONTROL(Wptr,RgnHnd)	Draws only the controls in Update Region
CALL	DRAW1CONTROL(Chndl)	Draw a single control if Visable

EXAMPLE

```
REM Control Manager Example Routines 5/10/87 A.G.
REM using the Control Manager toolbox routines
:
DIM T,L,B,R : T=10 : L=10 : B=160 : R=160 : REM Rectangle
WINDOW #1,"Control Example" : TEXT 4,12
FOR I=0 TO 2 : REM Make Control types 0 thru 2
  T=10 : B=40 : ID=I    : V=0 : GOSUB "Make Control"
  T=50 : B=90 : ID=I    : V=1 : GOSUB "Make Control"
  T=100: B=140: ID=I    :V=255: GOSUB "Make Control"
  T=150: B=190: ID=I+8 : V=1 : GOSUB "Make Control"
  L=L+165 : R=R+165
NEXT
T=200 :L=10 :B=232 :R=440 :ID=16 :V=25 : GOSUB "Make Control"
SBhndl&=Chndl&:T=240:L=10:B=270:R=120:ID=0  : V=1 : GOSUB "Make
Control"
ON TIMER(1) GOSUB "TIMER"
"LOOP"       : BREAK ON  : TIMER ON
GOTO "LOOP" : BREAK OFF : TIMER OFF
"TIMER" : Y=240
H=H+1 : IF H>255 THEN H=0 ELSE IF H=1 THEN H=255
CALL MOVECONTROL(Chndl&,X,Y) : X=X+10 : IF X>400 THEN X=10
CALL HILITECONTROL(Chndl&,H)
V=FN GETCTLVALUE(SBhndl&) : Title$="Scroll at"+STR$(V)
CALL SETCTITLE(Chndl&,Title$)
RETURN
"Make Control"
Title$="TYPE "+CHR$(48+ID)+",Value="+STR$(V)
Chndl&=FN NEWCONTROL(WINDOW(14),T,Title$,1,V,0,100,ID,0)
RETURN
```

MACINTOSH APPENDIX

MENU MANAGER

Terms used in this section:

Meta	Char	Description
;	....	Separates Multiple Items
^	Char	Specifies ICON Number (ASCII+208) ^1=ICON 257
!	Char	Check or Other Mark
<	Char	Special Char Style. Bold,Italic,Underline,Outline,Shadow
/	Char	Menu Key Equivalent
(		Disabled Menu Item
-		Dashed Line

mID	Word	Menu ID #: This is the Menu Number
Mttl	Str	Menu Title
Mhndl	Long	Menu Handle: Mhndl&=GETMHANDLE(Menu Number)
After	Word	Item # to Follow
Before	Word	ID of item to insert in front of
Mlist	Long	Handle to Complete Menu Bar
item	Word	Item # in Menu 0=Title, 1,2,3....

ASCII	Description
0	No Mark
11	Command Mark (Clover Leaf)
12	Check Mark
13	Diamond Mark for Menus
14	Apple Mark (Desk Accy's)

Mhndl=	FN NEWMENU(mID,Mttl)	Allocates Space for new Menu ID & Title
Mlist=	FN GETNEWMBAR(MbarID)	Gets Complete Resource Menu Bar of ID
Mlist=	FN GETMENUBAR	Saves Whole Menu in Mlist
Mhndl=	FN GETMENU(ID)	Get Resource Menu
Long=	FN MENUSELECT(Point)	Called if Mouse down in menu bar
Long=	FN MENUKEY(Char)	Returns Menu and item in long word
item=	FN COUNTMITEMS(Mhndl)	Returns # of items in the Menu
Mhndl=	FN GETMHANDLE(mID)	Returns Mhndl from mID
CALL	INITMENUS	*** Don't Use ***
CALL	DISPOSEMENU(Mhndl)	Dispose of Menu created with NewMenu
CALL	APPENDMENU(Mhndl,Str)	Add string of Menu(s) to End
CALL	ADDRESMENU(Mhndl,Type)	Add Resource Menu of type Example 'FONT'
CALL	INSERTRESMENU(Mhndl,Type,After)	Insert Resource Menu of Type After Item
CALL	INSERTMENU(Mhndl,Before)	Insert Menu Before ID specified
CALL	DRAWMENUBAR	Redraws Menu Bar (After Changes)
CALL	DELETEMENU(mID)	Deletes Menu With ID Specified
CALL	CLEARMENUBAR	Clears All Menus from Menu Bar
CALL	SETMENUBAR(Mlist)	Restores Whole Menu from Mlist
CALL	HILITEMENU(mID)	HighLites Menu mID,Un-Hilites if 0
CALL	SETITEM(Mhndl,item,Str)	Changes item to Str
CALL	GETITEM(Mhndl,item,Var$)	Gets Current item in Var$
CALL	DISABLEITEM(Mhndl,item)	Disables Menu item (Grays out)
CALL	ENABLEITEM(Mhndl,item)	Enables Menu item (Makes Selectable)
CALL	CHECKITEM(Mhndl,item,Byte)	Checks item if byte=True else Un-Checks

continued...

MACINTOSH APPENDIX 

continued from previous page...

CALL	SETITEMMARK(Mhndl,Item,Char)	Mark Menu item With Character
CALL	GETITEMMARK(Mhndl,Item,Var%)	Get Menu Mark into Var%
CALL	SETITEMICON(Mhndl,Item,Byte)	Put an ICON on item
CALL	GETITEMICON(Mhndl,Item,Var%)	Get the ICON number from item in Var%
CALL	SETITEMSTYLE(Mhndl,Item,Word)	Set the Character Style of item
CALL	GETITEMSTYLE(Mhndl,Item,Var%)	Get the Character Style of item in Var%
CALL	CALCMENUSIZE(Mhndl)	Internally Used
CALL	FLASHMENUBAR(mID)	Flash mID, If 0 Flash Whole Menu Bar
CALL	SETMENUFLASH(Cnt)	Number of Flashes after Selected

*****	128k ROM Only *****	
CALL	INSMENUITEM(Mhndl,Str,After)	Insert Menu item Str After item Specified
CALL	DELMENUITEM(Mhndl,item)	Delete Menu item specified

EXAMPLE

```
REM Display contents of Menu items
WINDOW 1,"Get Menu Items Example"
CLS: DIM A$(100)
Menu_handle&=FN GETMHANDLE (255)
LONG IF Menu_handle&
  N=FN COUNTMITEMS (Menu_handle&)
  FOR I=1 TO N
    CALL GETITEM (Menu_handle&,I,A$(I))
    PRINT CHR$(I+64)")  "A$(I)
  NEXT
END IF
END
```

MACINTOSH APPENDIX

TEXT EDIT ROUTINES

See example program under TEHANDLE in this appendix. The following terms are used in this section:

Drect	Rect	Destination Rectangle	(Fit Text in this Rect)
Vrect	Rect	View Rectangle	(Show text in this Rect)
TeHndl	Long	TEHandle same as ZBasic's TEHANDLE	
ChrHnd	Long	Handle to text Specified in edit record	
TxtPtr	Long	Pointer to 1st character of Text	
Len	Long	Length in Bytes	
Sels	Long	Select Start	
Sele	Long	Select End	
just	Word	0=Left,1=Center,-1=Right Text Justification	

---------- TeHandle Offsets ----------

0	DRect	DestRect	Destination Rectangle
8	VRect	ViewRect	View Recangle
16	SRect	SelRect	
24	word	LineHight	For Line Spacing
26	word	FontAscent	Caret/HighLighting Position
28	Point	SelPoint	
32	word	SelStart	Start of Selection
34	word	SelEnd	End of Selection
36	word	Active	
38	Proc	WordBreak	Proc Pointer (User Word Break)
42	Proc	ClikLoop	Proc Pointer (While Mouse Down)
46	Long	ClickTime	
50	word	ClickLoc	
52	Long	Carettime	
56	word	CaretState	
58	word	Just	justification
60	word	teLength	Length of Text (Up to 32767)
62	Hndl	hText	Handle to Text being edited
66	word	RecalBack	
68	word	recallines	
70	word	ClickStuff	
72	word	CrOnly	if < 0, New Line at <CR> only!
74	word	txFont	Font
76	word	txFace	Face
78	word	txMode	Mode
80	word	txSize	Size
82	Gptr	GrafPtr	GrafPort
86	Proc	HighHook	
90	Proc	caretHook	
94	word	nLines	Number of Lines
96	Words..	LineStarts	Array of Words Start of Each Line

continued....

MACINTOSH APPENDIX 

continued from previous page...

TeHndl=	FN TENEW(Drect,Vrect)	Get a Handle to New Text Edit Record
ChrHnd=	FN TEGETTEXT(TeHndl)	Returns Handle to text in edit record
CALL	TEINIT	*** Don't Use ***
CALL	TEDISPOSE(TeHndl)	Dispose of Memory allocated to Text Rec
CALL	TESETTEXT(TxtPtr,Len,TeHndl)	Makes a Copy of Text For Editing
CALL	TEIDLE(TeHndl)	Call this enough to keep caret Blinking
CALL	TECLICK(Point,Byte,TeHndl)	Call if Mouse Down in edit Field
CALL	TESETSELECT(Sels,Sele,TeHndl)	Change the Selection range
CALL	TEACTIVATE(TeHndl)	Activate an Edit Field
CALL	TEDEACTIVATE(TeHndl)	DeActivate an Edit Field
CALL	TEKEY(Char,TeHndl)	Send a Key to Text edit Field
CALL	TECUT(TeHndl)	Tell Text edit to Cut current selection
CALL	TECOPY(TeHndl)	Tell Text edit to Copy current selection
CALL	TEPASTE(TeHndl)	Tell Text edit to Paste current selection
CALL	TEDELETE(TeHndl)	Tell Text edit to Delete current selection
CALL	TEINSERT(TxtPtr,Len,TeHndl)	Insert Text just Before Selection Range
CALL	TESETJUST(just,TeHndl)	Change or Set Text justification
CALL	TEUPDATE(Rect,TeHndl)	Update Text within givin Rectangle
CALL	TEXTBOX(TxtPtr,Len,Rect,just)	Draws specified text in rect with just.
CALL	TESCROLL(dx,dy,TeHndl)	Scroll Text by dx,dy in view rectangle
CALL	TECALTEXT(TeHndl)	ReCalculate all Line Starts after Change

*****	128k ROMS Only	*****
CALL	TEAUTOVIEW(Auto,TeHndl)	If Auto=True Enables Auto Scroll
CALL	TESELVIEW(TeHndl)	If Auto Scroll Enabled,Makes Sel Visable
CALL	TEPINSCROLL(dx,dy,TeHndl)	Stops scrolling when last line in Vrect

MACINTOSH APPENDIX

DIALOG MANAGER

The following terms are used in this section:

Rproc	Long	Resume Procedure (After Syserr)
Dptr	Long	Dialog Pointer
BWPtr	Long	Behind which window pointer (-1 means in front of all)
Dsto	Long	Storage area for Dialog (Nil if in Heap)
title	Str	Title of Dialog Box
Visable	Byte	Visable if True
ProcID	Word	Dialog procedure ID (Like Window)
GoAway	Byte	True if GoAway Box
items	Handle	Handle to a dialog item list
dID	Word	Dialog Resource ID number
Fproc	Long	Filter procedure for Dialog (0 if none)
EvRec	Var	Event Record
aID	Word	Alert Template ID number
Sproc	Long	Error Sound Procedure
Ihndl	Long	Item Handle
strt	Word	Start select range
end	Word	end Select Range

Dptr=	FN NEWDIALOG(Dsto,Rect,title,Visable,ProcID,BWptr,GoAway,RefCon,items)	
Long=	FN GETNEWDIALOG(dID,Dsto,BWptr)	Get a resource defined Dialog Definition
Byte=	FN ISDIALOGEVENT(EvRec)	Returns True if Handled else DialogSelect
Byte=	FN DIALOGSELECT(EvRec,Dptr&,item%)	True if Enabled Hit,Else False
item=	FN ALERT(aID,Fproc)	Display & Handle Alert,Returns item hit
item=	FN STOPALERT(aID,Fproc)	Display & Handle Alert,Returns item hit
item=	FN NOTEALERT(aID,Fproc)	Display & Handle Alert,Returns item hit
item=	FN CAUTIONALERT(aID,Fproc)	Display & Handle Alert,Returns item hit
CALL	INITDIALOGS(Rproc)	**** Don't Use ****
CALL	ERRORSOUND(Sproc)	Set Special Error Sound Procedure
CALL	CLOSEDIALOG(Dptr)	Close Dialog Box
CALL	DISPOSDIALOG(Dptr)	Calls Close and Releases Memory
CALL	COULDDIALOG(dID)	Makes Dialog ID resource UnPurgable
CALL	FREEDIALOG(dID)	Makes Dialog ID resource Purgable
CALL	MODALDIALOG(Fproc,item%)	Returns Dialog item hit in item%
CALL	DRAWDIALOG(Dptr)	Just Draws the Dialog Box
CALL	COULDALERT(dID)	Makes Dialog ID resource UnPurgable
CALL	FREEALERT(dID)	Makes Dialog ID resource Purgable
CALL	PARAMTEXT(Str1,Str2,Str3,Str4)	Sets Param text For Dialog Boxes
CALL	GETDITEM(Dptr,item,Itype%,Ihndl&,Rect)	Returns Item Type,Handle,Rect
CALL	SETDITEM(Dptr,item,Type,Ihndl,Rect)	Sets item Type,Handle,Rect
CALL	GETITEXT(Ihndl,text$)	Returns item text from item Handle
CALL	SETITEXT(Ihndl,text)	Set text into item specified by Handle
CALL	SELITEXT(Dptr,item,Strt,end)	Set dialog start/end selection range
*****	128k Roms *****	
Item=	FN FINDDITEM(Dptr,Point)	Returns item number at Point specified
CALL	HIDEDITEM(Dptr,item)	Move item off screen, Invisable
CALL	SHOWDITEM(Dptr,item)	Bring back on screen, Visable
CALL	UPDTDIALOG(Dptr,RgnHnd)	Update area of dialog specified by Region

See Example on next page...

MACINTOSH APPENDIX

DIALOG MANAGER continued...

EXAMPLE1

Dialog Manager Item Types

Item	Description	Item	Description	ID	System Icons
1	ok Button	8	Static Text	0	Stop Icon
2	Cancel Button	16	Edit text (Dialogs Only)	1	Note Icon
4	Button	32	Icon	2	Caution Icon
5	Check Box	64	Pict Item		
6	Radio Button	0	User Item		
7	Rsrc Control	128	Add to Disable		

```
REM Returns all the item types from a dialog
REM See chart above
DIM itemType%(63),itemhndl&(63),3 iRect$(63)
X&=FN GETNEWDIALOG(135,0,-1) : item=0
DO : item=item+1
  CALL
  GETDITEM(X&,item,itemType%(item),itemhndl&(item),iRect$(item))
UNTIL itemhndl&(item)=0 OR item=63
CALL MODALDIALOG(0,X%)
CALL DISPOSDIALOG(X&)
PRINT X%
FOR I=1 TO item
  PRINT I,itemType%(I),itemhndl&(I)
NEXT
END
```

EXAMPLE2

```
REM Displays all resource dialogs found in
REM open resources (DA's, System and Appl dialogs etc.)
:
Type&=CVI("DLOG")
ResCount%=FN COUNTRESOURCES(Type&)
FOR J=1 TO ResCount%
  TRON X
  Hndl&=FN GETINDRESOURCE(Type&,J)
  CALL GETRESINFO(Hndl&,ID%,RType&,Name$)
  DlgPtr&=FN GETNEWDIALOG(ID%,0,-1)
  CALL DRAWDIALOG(DlgPtr&)
  DELAY 500
  CALL DISPOSDIALOG(DlgPtr&)
NEXT J
STOP
```

continued...

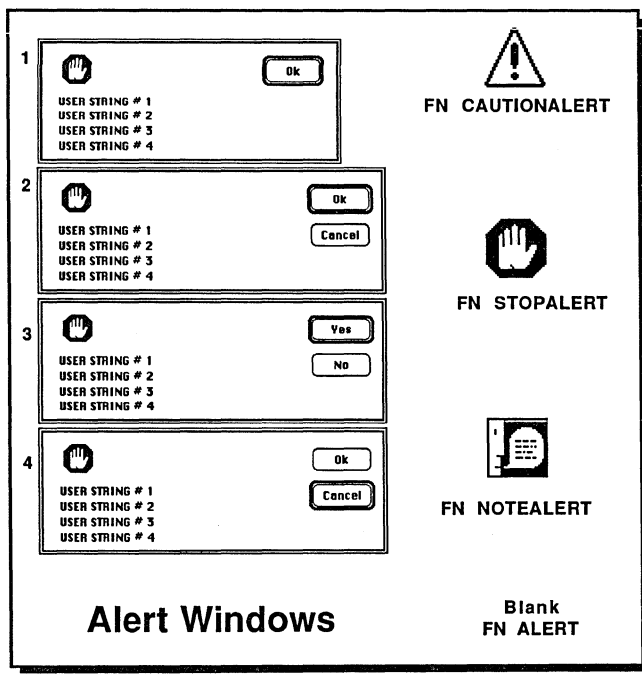
MACINTOSH APPENDIX

DIALOG continued....

EXAMPLE3 ZBasic allows you to use Macintosh™ CAUTION, NOTE and STOP windows easily.

```
DIM 63 A$(4)
FOR I = 1 TO 4
  A$(I)="USER STRING #"+STR$(I)
NEXT
CALL PARAMTEXT(A$(1), A$(2), A$(3), A$(4))
FOR I=1 TO 4
  REM Change next line to: ALERT, CAUTIONALERT or NOTEALERT
  REM for the ICON needed in your application.
  Response% = FN STOPALERT (I,0)
NEXT
```

Response% returns the users response. If RETURN is pressed, the highlighted button is assumed to be the answer. Note that the answers are always 1 or 2 with the highlighted button always number one:



MACINTOSH APPENDIX

DESK ACCESSORIES

Terms used in this section:

ID%=	FN OPENDESKACC(Name)	Open DA with name in 'Name'
CALL	CLOSEDESKACC(ID%)	Close open DA with ID%

EXAMPLE1

```
WINDOW 1,"DESK ACCESSORY.BAS"
CLS
DIM A$(100)
:
Menu_handle&=FN GETMHANDLE (255)
:
LONG IF Menu_handle&
  N=FN COUNTMITEMS (Menu_handle&)
  FOR I=1 TO N
    CALL GETITEM (Menu_handle&,I,A$(I))
    PRINT CHR$(I+64)")   "A$(I)
  NEXT
END IF
:
PRINT
INPUT&1,;! "DA NUMBER TO OPEN  - X = QUIT";DA$
:
CLS
DA$=UCASE$(DA$)
IF DA$ = "X" THEN END
DA$=A$(ASC(DA$)-64)
PRINT
X=FN OPENDESKACC (DA$)
WINDOW OUTPUT 1
END
```

EXAMPLE2

```
WINDOW 1,"Enable/Disable Desk Accessories"
:
"DISABLE DA"
Var&=FN GETMHANDLE(255)
CALL DISABLEITEM(Var&,0)
:
"ENABLE DA"
CALL ENABLEITEM(Var&,0)
```

MACINTOSH APPENDIX

STRING HANDLING ROUTINES

Terms used in this section:

StrHnd	Long	Handle to Relocatable String

CALL	SETSTRING(StrHnd,Str$)	Makes Str from StrHnd
StrHnd=	FN NEWSTRING(Str)	Makes StrHnd from Str
StrHnd=	FN GETSTRING(ID)	Returns StrHnd for Resource ID
Long=	FN MUNGER(Long,Long,Var,Long,Var,Long)	Manipulates Bytes in Strings
Word=	FN IUMAGSTRING(Var,Var,Word,Word)	Compares 2 strings
Word=	FN IUMAGIDSTRING(Var,Var,Word,Word)	Compares 2 Strings

INTERNATIONAL UTILITIES

Terms used in this section:

Second	Long	Long Word of Seconds since January 1, 1904
Form	Word	Mask for Form
Date$	Var$	String Variable to store Date String
Time$	Var$	String Variable to store Time String
IID	Word	International Resource ID 0,1
IRhndl	Long	International Resource Handle
itlprm	Long	International Parameter

CALL	INITPACK(PackID)	Init Package
CALL	INITALLPACKS	*** Don't Use *** init all pkg's
Byte=	FN IUMETRIC	True if Metric System Used
IRhndl=	FN IUGETINTL(IID)	Returns Handle to itnl Resource
CALL	IUDATESTRING(Second,Form,date$)	String Date from Seconds
CALL	IUDATEPSTRING(Second,Form,date$,IRhndl)	Returns Date
CALL	IUTIMESTRING(Second,Byte,Time$)	Returns Time$,if True Seconds
CALL	IUTIMEPSTRING(Second,Byte,Time$,IRhndl)	Returns Time$,If true Seconds
CALL	IUSETINTL(Refnum,ID,itlprm)	Sets the Itnl rsrc to Specified

MACINTOSH APPENDIX 

FILE INPUT/OUTPUT

Terms used in this section:

Drive	Word	Word specifing Which Drive
EvMsg	Long	Event Message Err Code in High Word
Prmpt	Str	Prompt String
Dproc	Long	Dialog Hook Procedure
Srply	Var	Standard File Reply
dlD	Word	Dialog ID
Fproc	Long	Filter Procedure
FFproc	Long	File Filter Procedure
Ntypes	Word	Number of File Types
TypList	Var	Pointer to File Type List

Word=	FN DIBADMOUNT(Point,EvMsg)	Call After a disk insert Event
Word=	FN DIFORMAT(Drive)	Format Disk in Drive Specified
Word=	FN DIVERIFY(Drive)	Verify Disk in Drive specified
Word=	FN DIZERO(Drive,Name)	Initialize Disk in Drive,Vol Name
CALL	DILOAD	Load and Make Disk Init Pkg UnPurgable
CALL	DIUNLOAD	UnLoad and Make Disk Init Pkg Purgable

Standard Open New File Box Procedures

```
CALL    SFPUTFILE(Point,Prmpt,Name,Dproc,Srply)
CALL    SFPPUTFILE(Point,Prmpt,Name,Dproc,Srply,dlD,Fproc)
```

Standard Open Old File Box Procedures

```
CALL    SFGETFILE(Point,Prmpt,FFproc,Ntypes,TypList,Dproc,Srply)
CALL    SFPGETFILE(Point,Prmpt,FFproc,Ntypes,TypList,Dproc,Srply,dlD,Fproc)
```

MACINTOSH APPENDIX

MOUSE

Byte=	FN BUTTON	Current Mouse Button Status 0=Up
Byte=	FN STILLDOWN	True if Mouse not let up Since Last
Byte=	FN WAITMOUSEUP	Same as StillDown Except No Mouse Up
CALL	GETMOUSE(Point)	Returns Mouse in Local Coordinates

EXAMPLE

```
REM VARIABLES A,D,W,L = FONTINFO; SEE
REM   PH = HORIZONTAL PIXEL COORDINATE
REM   PV = VERTICAL PIXEL COORDINATE
REM   CH = HORIZONTAL CURSOR COORDINATE
REM   CV = VERTICAL CURSOR COORDINATE
REM   DH = DIFFERENCE IN HORIZONTAL COORDINATES SINCE LAST
GETMOUSE
REM     DV = DIFFERENCE IN VERTICAL COORDINATES SINCE LAST
GETMOUSE
TRONB: COORDINATE WINDOW:CLS:TEXT ,,,0
DIM A,D,W,L:O&=0
DO
  CALL BACKCOLOR (A)
  CALL GETWMGRPORT (X&)
  D&=FN DELTAPOINT (X&,O&):O&=X&
  H=A+D+L
  PH=X& AND 65535
  PV=X&>>16
  CV=PV/H
  CH=PH/W
  DV=D&>>16
  DH=D& AND 65535
  PRINT@(0,10)"PIXEL  COORDINATES  ";PH",";PV
  PRINT@(0,12)"CURSOR COORDINATES  ";CH",";CV
  :
  LONG IF PV<0 OR PH<0 OR PH>WINDOW(6) OR PV>WINDOW(7)
    PRINT@(0,14);"** OFF SCREEN **"
  XELSE
    PRINT@(0,14);"** ON  SCREEN **"
  END IF
  :
  LONG IF DH<>0 OR DV<> 0
    PRINT@(0,18)"SINCE LAST TIME     ";DH","DV;"        "
  END IF
  :
  LONG IF FN BUTTON
    PRINT@(0,16);"MOUSE DOWN"
  XELSE
    PRINT@(0,16);"MOUSE UP   "
  END IF
UNTIL LEN(INKEY$): END
```

NOTE: Also see example program under "Control Manager".

MACINTOSH APPENDIX

MISCELLANEOUS

Terms used in this section:

Keys	8 Bytes	Current Keyboard BitMap 0=Up,1=Down Bits

Word=	FN RANDOM	Returns Uniform Random #-32768 to 32767
Long=	FN TICKCOUNT	Returns # of Ticks (1/60 Second)
CALL	GETKEYS(Keys)	Returns Current Keyboard Status of ALL keys
CALL	SYSBEEP(Word)	System Beep for Duration Word/60 seconds
CALL	SYSTEMTASK	Allows Desk Accy's Update time (Clock etc.)

EXAMPLE

```
REM Displays ALL keys; including Command, Shift and Option keys!
DIM A%(7),B%(7): CLS : FOR I=0 TO 7 : B%(I)=-1 : NEXT
"KEYSCAN"
  CALL GETKEYS(A%(0)): LOCATE 0,0
  FOR I=0 TO 7
    LONG IF B%(I)<> A%(I)
      LOCATE 0,I : CLS LINE : PRINT I,BIN$(A%(I)) : B%(I)=A%(I)
    ENDIF
  NEXT
  TRONX:REM Press <Command Period> to stop
GOTO "KEYSCAN"
```

Byte=	FN CHECKUPDATE(Var)	See Inside Mac
Byte=	FN SYSTEMEDIT(Word)	See Inside Mac
Byte=	FN SYSTEMEVENT(Var)	See Inside Mac
Byte=	FN GETNEXTEVENT(Word,Var)	See Inside Mac
Byte=	FN EVENTAVAIL(Word,Var)	See Inside Mac
CALL	SYSTEMCLICK(Var,Long)	See Inside Mac
CALL	SYSTEMMENU(Long)	See Inside Mac

CLIPBOARD SCRAP MANAGER

Note: This section refers to the Clipboard; NOT the scrapbook. Terms used in this section:

Sptr	Long	Pointer to Scrap Info
Dhnd	Long	Destination Handle for Scrap
Ofst	Long	Offset to Scrap
Slen	Long	Length of scrap in Bytes
Type	Long	Scrap Type Example: 'TEXT' or 'PICT'
Sptr	Long	Pointer to Source for Scrap

---------- Scrap Info Record ----------

0	Long	Scrap Size in bytes
4	Long	Handle to Desk Scrap
8	Word	Count Changed by 'ZeroScrap'
10	Word	Scrap State,+=In Memory,0=On Disk,-=Not Initialized
12	StrPtr	Scrap File Name

Sptr=	FN INFOSCRAP	Get Pointer to Scrap Info
OSerr=	FN UNLOADSCRAP	UnLoad Scrap in Memory to Disk
OSerr=	FN LOADSCRAP	Read Scrap from Disk Into Memory
Slen=	FN GETSCRAP(Dhnd,Type,ofst&)	Get current Scrap from Disk or Memory
OSerr=	FN ZEROSCRAP	Clear current scrap contents
OSerr=	FN PUTSCRAP(Slen,Type,Sptr)	Puts Data to Scrap in Memory or on Disk

MACINTOSH APPENDIX

CALCULATIONS

Terms used in this section:

Numer	Word	Numerator
Denom	Word	Denominator
Angle	Word	0 to 360 Degrees
Slope	Long	High Word=dy, Low Word=dx

Sptr=	FN FIXRATIO(Numer,Denom)	Fixed Point Divide of two Integers
Long=	FN FIXMUL(Fixp,Fixp)	Fixed Point Multiply of to FIxed Point
Word=	FN FIXROUND(Fixed)	Rounds +Fixp Number to Nearest Integer
CALL	LONGMUL(Long,Long,Var)	Multiply 2 Longs to a 64 bit Result at Var
Point=	FN DELTAPOINT(Point,Point)	Compute Difference of 2 x,y Points
Slope=	FN SLOPEFROMANGLE(Angle)	Compute Delta x,y Point From Angle
Angle=	FN ANGLEFROMSLOPE(Slope)	Compute Angle from Delta x,y Point
CALL	PTTOANGLE(Rect,Point,Angle%)	Calc Integer Angle to Center of Rect

BINARY CALCULATIONS

Terms used in this section:

Bit	Long	Bit # to Set/Reset or Test 0 to 4,294,967,295

Byte=	FN BITTST(Var,Bit)	Tests a Bit offset Var with Long bit number
CALL	BITSET(Var,Bit)	Set a Bit Offset Var with Long bit number
CALL	BITCLR(Var,Bit)	Reset a Bit Offset Var with Long bit number
Long=	FN BITAND(Long,Long)	AND's two Long words to result
Long=	FN BITOR(Long,Long)	OR's two Long words to Result
Long=	FN BITXOR(Long,Long)	XOR's two Long words to Result
Long=	FN BITNOT(Long)	Returns the Logical NOT of long word
Long=	FN BITSHIFT(Long,Cnt)	Shifts the Long word Cnt Bits +/-
Word=	FN HIWORD(Long)	Returns the High Word of a long Word
Word=	FN LOWORD(Long)	Returns the Low word of a Long Word

PACKBITS AND UNPACKBITS

Terms used in this section:

SrcPtr	Var&	Variable which contains Source Address
DstPtr	Var&	Variable which contains Destination Address
SBytes	Word	Source Bytes (Normally 72 for MacPaint,127 Max)

Note SrcPtr & DestPtr Addresses Updated after ToolBox.

CALL	PACKBITS(SrcPtr,DstPtr,SBytes)	Pack Source bytes to Destination
CALL	UNPACKBITS(SrcPtr,DstPtr,SByts)	UnPack Source to Destination Bytes

Note: See example of these two calls under READ FILE# which loads and saves MacPaint images.

MACINTOSH APPENDIX

SCUZZY FUNCTIONS (128K ROMS ONLY)

Terms used in this section:

*******	SCSI Functions	*** 128k ROMS or Greater ***
OSerr	Word	Operating System Error Code
tID	Word	Target SCSI ID
VCmd	Var	Command Source of Bytes
Wait	Long	Number of ticks (timeout)
TIB	Var	Transfer Instruction Block
STA%	Var%	Integer Variable Status
MSG%	Var%	Integer Variable Message

------- TIB Block ----------		-- Transfer Inst OpCodes --		
0	scOpCode	Integer	1	SCINC
2	scParam1	Long	2	SCNOINC
6	scParam2	Long	3	SCADD
4	SCMOVE			
5	SCLOOP			
6	SCNOP			
7	SCSTOP			
8	SCCOMP			

---------- ScsiB bits ----------

15	14	13	12	11	10	9	8
EndDma	DmaReq	PtyErr	IntReq	PhsMat	BsyErr	Atn	Ack
7	6	5	4	3	2	1	0
Rst	Bsy	Req	Msg	C/D	I/O	Sel	DBP

OSerr=	FN SCSIRESET	Resets the SCSI Buss
OSerr=	FN SCSIGET	Arbitrates for Use of SCSI bus
OSerr=	FN SCSISELECT(tID)	Selects the Device tID
OSerr=	FN SCSICMD(VCmd,Cnt)	Sends the Command at VCmd,Cnt Bytes
OSerr=	FN SCSICOMPLETE(STA%,MSG%,Wait)	Gives the current Command Wait Ticks
OSerr=	FN SCSIREAD(TIB)	Transfers Data from Target to Initiator
OSerr=	FN SCSIWRITE(TIB)	Transfers Data From Initiator to Target
OSerr=	FN SCSIINSTALL	
OSerr=	FN SCSIRBLIND(TIB)	Same as Write, Does not Poll /REQ
OSerr=	FN SCSIWBLIND(TIB)	Same as Read, Does not Poll /REQ
ScsiB=	FN SCSISTAT	Returns BitMap of SCSI Control/Status

MACINTOSH APPENDIX

LIST MANAGER (128K ROMS ONLY)

Terms used in this section:

lhndl	Long	Handle to List Record
Cell	Point	Cell Description
vCell	var	Cell Variable
Lsize	Point	Cell Size in Pixels
Vrect	Rect	View Rectangle
Drect	Rect	Data Rectangle
pID	ID	Proc ID
Drawit	Byte	Draw it Flag
Grow	Byte	Has Grow box Flag
Hscrl	Byte	Has Horzontal Scroll Bar Flag
Vscrl	Byte	Has Vertical Scroll Bar Flag
Cnum	Word	Column Number
Rnum	Word	Row Number
Dptr	Long	Pointer to Data
Dlen	word	Length of data in bytes
Sproc	long	Search Procedure
hNext	byte	Flag for next horz Cell
vNext	byte	Flag for next Vert Cell
Doff	Word	Data offset in bytes
Lwidth	Word	List Width in pixels
Lhight	Word	list Hight in Pixels
dCol	Word	Number of Columns
dRow	Word	Number of Rows

---------- List Record ----------

Offset	Size	Name	Description
0	Rect	rView	List's Display Rectangle
8	Gptr	port	List's grafPort
12	Point	indent	Indent Distance
16	Point	cellSize	Cell Size
20	Rect	visable	Boundry of Visable Cells
28	Chndl	vScroll	Vert Scroll Handle
32	Chndl	hScroll	Horz Scroll Handle
36	Byte	selFlags	Selection Flags
37	Byte	lActive	Boolean True if Active
38	Byte	lReserved	Resurved
39	Byte	listFlags	ListFlags Auto-Scroll Flags
40	Long	clikTime	Click-Time Time of Last Click
44	Point	clikLoc	Position of last click
48	Point	MouseLoc	Current Mouse Location
52	Ptr	lClickLoop	Routine for Lclick
56	Cell	lastClick	Last Click,Last Cell Clicked
60	Long	RefCon	RefCon
64	Hndl	listDefHandle	List's Definition Procedure
68	Hndl	usrHandle	User Handle
72	Rect	dataBounds	Boundry of Cells Allocated
76	Word	maxIndex	MaxIndex (Used Internally)
78...	Words..	cellArray	Cell Array of integer Offsets to Data

continued....

MACINTOSH APPENDIX 

LIST MANAGER (128K ROMS ONLY) continued

---------- selFlags Bit Definitions ----------

Bit	Value	Name	Description
0	1	????	????
1	2	INoNilHilite	1 = don't hilite empty cells
2	4	IUseSense	1 = shift should use sense of start cell
3	8	INoRect	1 = don't grow (shift,drag) selection as rect
4	16	INoExtend	1 = don't extend shift selections
5	32	INoDisjoint	1 = turn off selections on click
6	64	IExtendDrag	1 = drag select without shift key
7	128	IOnlyOne	0 = multiple selections, 1 = one

---------- listFlags Bit Definitions ----------

Bit	Value	Name	Description
0	1	IDoHAutoscroll	1 = allow horizontal autoscrolling
1	2	IDoVAutoscroll	1 = allow vertical autoscrolling

Lhndl=	FN LNEW(Vrect,Drect,Lsize,pID,Wptr,Drawit,Grow,Hscrl,Vscrl)	New List!
Word=	FN LADDCOLUMN(Cnt,Cnum,Lhndl)	Add Cnt Columns starting at Cnum
Word=	FN LADDROW(Cnt,Rnum,Lhndl)	Add Cnt Rows starting ar Rnum
Byte=	FN LGETSELECT(Byte,Cell,Lhndl)	Return True if Selected or Cell coord
Byte=	FN LCLICK(Point,Word,Lhndl)	Call if Mouse down in Dest Rect
Long=	FN LLASTCLICK(Lhndl)	Returns Cell coord of Last Cell Clicked
Byte=	FN LNEXTCELL(hNext,vNext,vCell,Lhndl)	True if Next Cell in vCell
Byte=	FN LSEARCH(Dptr,Dlen,Sproc,vCell,Lhndl)	True if Found & Cell in vCell
CALL	LDISPOSE(Lhndl)	Dispose of List
CALL	LDELCOLUMN(Cnt,Cnum,Lhndl)	Delete Cnt Columns starting at Cnum
CALL	LDELROW(Cnt,Rnum,Lhndl)	Delete Cnt Rowa starting at Rnum
CALL	LADDTOCELL(Dptr,Dlen,Cell,Lhndl)	Appends Data to Cell
CALL	LCLRCELL(Cell,Lhndl)	Clear Contents of Cell
CALL	LGETCELL(Dptr,Dlen%,Cell,Lhndl)	Gets Data in Cell to Dptr and Dlen%
CALL	LSETCELL(Dptr,Dlen,Cell,Lhndl)	Sets Data for Dlen into Cell Specified
CALL	LCELLSIZE(Cell,Lhndl)	Sets the CellSize Field before Drawing
CALL	LSETSELECT(Byte,Cell,Lhndl)	Select or de-Select if Byte True/False
CALL	LFIND(Doff%,Dlen%,Cell,Lhndl)	Returns offset & Length of Cell Data
CALL	LRECT(Rect,Cell,Lhndl)	Returns in Local coord Cell Rectangle
CALL	LSIZE(Lwidth,Lhight,Lhndl)	Adjust List & Scrolls after SizeWindow
CALL	LDRAW(Cell,Lhndl)	Redraw Cell Specified
CALL	LDODRAW(Byte,Lhndl)	False Disables cell Drawing, True Enables
CALL	LSCROLL(dCol,dRow,Lhndl)	Scrolls the List Number Rows/Columns
CALL	LAUTOSCROLL(Lhndl)	Scroll List til Selected Cell top/Left
CALL	LUPDATE(RgnHnd,Lhndl)	Update Rgn Area of Display
CALL	LACTIVATE(Byte,Lhndl)	True:Activate List, False De-Activate

continued...

MACINTOSH APPENDIX

LIST MANAGER (128K ROMS ONLY) continued

EXAMPLE

```
REM  Example of using the LIST-MANAGER Toolbox Routines
:
COORDINATE WINDOW : REM Use Window Coordinates
DIM T,L,B,R : T=0 : L=0 : B=8192 : R=8192 : REM Big Rectangle
DIM VT,VL,VB,VR,DT,DL,DB,DR,LH,LW : REM Two Rectangles
DIM WHAT,MSG&,WHEN&,WHERE&,MDF : REM Event Message Variable
DIM CELLY,CELLX : REM Cell Variable Definition
WINDOW #1 : CALL CLIPRECT(T) : REM Window For List Manager
VT=0 : VL=0 : VB=WINDOW(7)-1 : VR=WINDOW(6)-1 : REM VIEW RECTANGLE
DT=0 : DL=0 : DB=20 : DR=20 : REM DEST RECTANGLE
CELLY=20 : CELLX=90
LHNDL&=FN LNEW(VT,DT,CELLY,0,WINDOW(14),-1,-1,-1,-1)
FOR CELLY=0 TO 19
  FOR CELLX=0 TO 19
    A$="*TEST*"+STR$(CELLY)+STR$(CELLX)
    CALL LSETCELL(VARPTR(A$)+1,LEN(A$),CELLY,LHNDL&)
  NEXT
NEXT
DO
  X=FN GETNEXTEVENT(2+4+8+16+32+64,WHAT)
  ON WHAT GOSUB "MOUSE DOWN","MOUSE UP","KEY DOWN","KEY UP","KEY
DOWN","UPDATE"
UNTIL 0
"MOUSE DOWN"
WHY=FN FINDWINDOW(WHERE&,WPTR&)
LONG IF WHY=3 AND WPTR&=WINDOW(14)
  CALL GLOBALTOLOCAL(WHERE&)
  X=FN LCLICK(WHERE&,MDF,LHNDL&)
END IF
RETURN
"KEY DOWN"
LONG IF (MDF AND 256)<>0 AND (MSG& AND 255)=ASC(".")
  CALL LDISPOSE(LHNDL&)
  STOP
XELSE
  BEEP
ENDIF
RETURN
"KEY UP"
"MOUSE UP"
RETURN
"UPDATE"
WPTR&=WINDOW(14)
CALL BEGINUPDATE(WPTR&)
CALL LUPDATE(PEEK LONG(WPTR&+24),LHNDL&)
CALL ENDUPDATE(WPTR&)
RETURN
END
```

MACINTOSH APPENDIX

USEFUL MEMORY LOCATIONS

Name	Size	Addr	Description
Acount	Word	&A9A	Alert Stage 0 to 3
ApFontID	Word	&984	Font Number of Application Font
ApplLimit	Long	&130	Application Heap Limit
ApplScratch	Byte12	&A78	Application Scratch Area 12 bytes
AppParmHandle	Long	&AEC	Handle to Finder Information
BootDrive	Word	&210	Which drive was system booted
CaretTime	Long	&2F4	Caret Blink time in ticks (1/60 second)
CurApName	Str31	&910	Current Application Name Len/text 32 bytes
CurApRefNum	Word	&900	Current Application Resource RefNum
CurrentA5	Long	&904	Value of Application A5 global pointer
DAStrings	Long4	&AA0	Handles to ParamText Strings 4 Long Words
DefltStack	Long	&322	Default space allotment for stack
DeskPattern	Byte8	&A3C	DeskTop Pattern
DlgFont	Word	&AFA	Font Number for Dialogs and Alerts
DoubleTime	Long	&2F0	Double Click Interval in Ticks
DragPattern	Byte8	&A34	Pattern for Outline For Dragging
DSAlertRect	Rect	&3F8	Rectangle for System Error Alert
FinderName	Str15	&2E0	Name of Finder (What Appl to Execute on Exit)
FScaleDisable	Byte	&A63	0=Enable, nz=Disable Font Scaling
MemErr	Word	&220	Last memory error
PortBUse	Byte	&291	Serial Port B Use Flag Byte
PrintErr	Word	&944	Result Code from last print manager Routine
ResErr	Word	&A60	Current Value of ResErr (Resource error Code)
RndSeed	Long	&156	Random Number Seed (For Toolbox Random #'s)
RomBase	Long	&2AE	Base Address of ROM
SCCrd	Long	&1D8	SCC read Base Address
SCCwr	Long	&1DC	SCC write Base Address
ScrapCount	Word	&968	Count Changed by ZeroScrap
ScrapHandle	Long	&964	Handle to Desk Scrap in Memory
ScrapName	Long	&96C	Pointer to scrap file name (Str255 format)
ScrapSize	Long	&960	Size in Bytes of Desk Scrap
ScrapState	Word	&96A	Tells where scrap is (Disk/Memory etc.)
ScrDmpEnb	Byte	&2F8	Cmd-Shift-Number Disable=0
ScrHRes	Word	&104	Pixels per Inch horizontal
ScrBase	Long	&824	Base Address of Screen
ScrVRes	Word	&102	Pixels per Inch Vertical
SdVolume	Byte	&260	Speaker Volume (Low 3 bits only)
SPAlarm	Long	&200	Alarm Setting (Seconds)
SPPortA	Word	&1FC	Modem Port Configuration
SPPortB	Word	&1FE	Printer Port Configuration
SPPrint	Byte	&207	Printer connection Byte
SysFontFAM	Word	&BA6	System Font number used in menus etc. (New Roms)
SysFontSize	Word	&Ba8	System font size used in meues etc. (New Roms)
TEScrpHandle	Long	&AB4	Handle to TEScrap
TEScrpLength	Word	&AB0	Length of TEScrap
Ticks	Long	&16A	Number of ticks since System Startup (1/60 second)
Time	Long	&20C	Number of Seconds since Jan 1st 1904
VIA	Long	&1DA	VIA base Address
WindowList	Long	&9DA	Pointer to 1st Window in Window List (0=no Wind)
WMgrPort	Long	&9DE	Pointer to Window Manager Port

MACINTOSH APPENDIX

ALPHABETICAL LISTING OF TOOLBOX TERMS

Term	Type	Definition	Page
$	String	Pointer to String Variable	162
&	Long Integer	Pointer to Long Variable	162
%	Integer	Pointer to Integer Variable	162
After	Word	Item number to Follow	181
alD	Word	Alert Template ID number	185
Angle	Word	0 to 360 Degrees	193
Angle	Word	Degrees (Integer 0 to 360)	165
Angle	Word	Number of Degrees ClockWise (0 to 360)	171
Aproc	Long	Pointer to Action Procedure	179
Attr	Word	File attributes (See "Inside Mac")	164
Axis	Word	1=Horz Move, 2=Vertical Move, 0=Both	179
Before	Word	ID of item to insert in front of	181
Bit	Long	Bit number to Set/Reset or Test 0 to 4,294,967,295	193
BitMap	Long	Pointer to BitMap information	165
BWPtr	Long	Behind which window pointer (-1 means in front of all)	185
BWptr	Long	Behind Which Window 'Wptr'	177
Byte	8 Bit	Boolean True <> 0 False = 0	161
Cell	Point	Cell Description	195
Char	Byte	Passed as Word with ASCII in both hi & low Bytes	161
CHndl	Long	Handle to cursor structure	166
Chndl	Long	Control Handle	179
ChrHnd	Long	Handle to text Specified in edit record	183
cID	Word	Contol Resource ID number	179
Cnum	Word	Column Number	195
Color	Long	Color to be used	166
Curs	Var	Cursor pattern: 68 total bytes	166
Data	Long	Handle to actual Data (Nil if No Data)	173
Date$	Var$	String Variable to store Date String	189
DBit	Long	Destination BitMap	175
dCol	Word	Number of Columns	195
Denom	Word	Denominator	193
Dhnd	Long	Destination Handle for Scrap	192
dID	Word	Dialog ID	190
dID	Word	Dialog Resource ID number	185
Dlen	word	Length of data in bytes	195
Doff	Word	Data offset in bytes	195
Dproc	Long	Dialog Hook Procedure	190
DPtr	Long	Destination Pointer	175
Dptr	Long	Dialog Pointer	185
Dptr	Long	Pointer to Data	195
Drawit	Byte	Draw it Flag	195
DRect	Rect	Defines the target Rectangle (same structure as Rect)	165
Drect	Rect	Data Rectangle	195
Drect	Rect	Destination Rectangle (Fit Text in this Rect)	183
DRgn	RgnHnd	Destination Region Handle	172
Drive	Word	Word specifing Which Drive	190
DRow	Word	Destination Row	175
dRow	Word	Number of Rows	195
Dsto	Long	Storage area for Dialog (Nil if in Heap)	185
DstPtr	Var&	Variable which contains Destination Address	193
dx	Word	Delta X Position	165
dxy	Long	Delta x,y (0 if no change)	177
dy	Word	Delta Y Position	165
end	Word	end Select Range	185
ErrNum	Word	Resource manager error number	164
EvMsg	Long	Event Message Err Code in High Word	190
EvRec	Var	Event Record	185
Face	Word	Text Face bits (bold, italic etc.)	176
FFproc	Long	File Filter Procedure	190
Fixp	Long	Fixed Point Example: F&=num#*65536	176
Fmod	Word	Text Mode (Mode 0 to 7)	176
Fname	Str	File name: "quotes" or Variable$	164
Fname$	Var$	File name in variable$	164

MACINTOSH APPENDIX

Fnum	Word	Text Font Number	176
Form	Word	Mask for Form	189
Fproc	Long	Filter Procedure	190
Fproc	Long	Filter procedure for Dialog (0 if none)	185
Fsize	Word	Font Size 1 to 127	176
globe	Long	Long Word Pointer to application globals	165
GoAway	Byte	True if GoAway Box	185
GrfPtr	Long	Pointer to space for GrafPort	165
Grow	Byte	Has Grow box Flag	195
gWay	Byte	True if GoAway Box	177
h	Word	Horizontal Position	174
Hght	Word	Hight (in Pixels)	175
Hndl	Long	FirstByte= PEEK(PEEK LONG(Hndl))	163
hNext	byte	Flag for next horz Cell	195
Horz	Word	Size in Pixels Width	179
Hscrl	Byte	Has Horzontal Scroll Bar Flag	195
ID	Word	ID number of resource	164
Ihndl	Long	Item Handle	185
IID	Word	International Resource ID 0,1	189
Indx	Word	Index number (From 1 to n typically)	164
IRhndl	Long	International Resource Handle	189
item	Word	Item # in Menu 0=Title, 1,2,3....	181
items	Handle	Handle to a dialog item list	185
itlprm	Long	International Parameter	189
just	Word	0=Left,1=Center,-1=Right Text Justification	183
Keys	8 Bytes	Current Keyboard BitMap 0=Up,1=Down Bits	192
Kind	Word	Type of Comment(Comment Number)	173
Len	Long	Length in Bytes	183
Lhight	Word	list Hight in Pixels	195
lhndl	Long	Handle to List Record	195
Lrect	Rect	Limit Rectangle	179
Lsize	Point	Cell Size in Pixels	195
Lwidth	Word	List Width in pixels	195
Max	Word	Maximum control Value	179
MBit	Long	Mask BitMap	175
Mhndl	Long	Menu Handle: Mhndl&=GETMHANDLE(Menu Number)	181
mID	Word	Menu ID number: This is the Menu Number	181
Min	Word	Minumum control Value	179
Mlist	Long	Handle to Complete Menu Bar	181
MRect	Var	Mask Rectangle	175
MSG%	Var%	Integer Variable Message	194
Mttl	Str	Menu Title	181
Ntypes	Word	Number of File Types	190
Numer	Word	Numerator	193
Offset	Type	Description	166
Offset	Type	Description	166
Ofst	Long	Offset to Scrap	192
Oh	Word	Height of rounded edge in Pixels	170
OSerr	Word	Operating System Error Code	194
Ow	Word	Width of rounded edge in Pixels	170
Pat	Var	Pattern Definition 8 bytes (Bits:1=Black)	166
PenRec	Var	PenState Record 18 bytes	166
PHndl	Long	Handle to pattern of 8 Bytes	166
PicHnd	Long	Handle to Picture(PICTURES are limited to 32K)	173
pID	ID	Proc ID	195
pID	Word	Proc ID, Window Definition	177
PlyHnd	Long	Handle to Poly	174
Pmode	Word	Pattern Transfer Mode	166
Point	Var	Pointer to the first byte of two "Words" specifying Y, X	165
Point	Var	Var$, Var& or Var%,Var%	162
Prmpt	Str	Prompt String	190
Proc	Long	Pointer to a Machine Language Routine	162
ProcID	Word	Control Definition Procedure ID	179
ProcID	Word	Dialog procedure ID (Like Window)	185
Ptr	Long	FirstByte= PEEK(Ptr)	163
Pxls	Word	Number of Pixels	176

MACINTOSH APPENDIX

Rect	Var	Pointer to the first byte of four "Words"	165
RefCon	Long	Controls Ref Value	179
RefCon	Long	Window Long word Ref Value	177
RefNum	Word	Resource file reference number	164
RgnHnd	Long	Handle to a Region	165
Rhndl	Long	Handle to resource	164
Rnum	Word	Row Number	195
Rproc	Long	Resume Procedure (After Syserr)	185
Rsize	Long	Resource size in bytes	164
SBit	Long	Source BitMap	175
SBytes	Word	Source Bytes (Normally 72 for MacPaint,127 Max)	193
Second	Long	Long Word of Seconds since January 1, 1904	189
Seedh	Word	Vertical Offset in Pixels	175
SeedV	Word	Horizontal Offset in Pixels	175
Sele	Long	Select End	183
Sels	Long	Select Start	183
Size	Word	number of Bytes (0 if No Data)	173
Slen	Long	Length of scrap in Bytes	192
Slope	Long	High Word=dy, Low Word=dx	193
Slopr	Rect	Slop Rectangle	179
Smode	Word	Source Transfer Mode	166
Spos	Word	Start offset position in bytes	165
Sproc	Long	Error Sound Procedure	185
Sproc	long	Search Procedure	195
SPtr	Long	Source Pointer	175
Sptr	Long	Pointer to Scrap Info	192
Sptr	Long	Pointer to Source for Scrap	192
SrcPtr	Var&	Variable which contains Source Address	193
SRect	Rect	Defines the source Rectangle (same structure as Rect)	165
SRgn	RgnHnd	Source Region Handle	172
SRow	Word	Source Row	175
Srply	Var	Standard File Reply	190
STA%	Var%	Integer Variable Status	194
Str	Len+String	Either a quoted string or a string variable	161
StrHnd	Long	Handle to Relocatable String	189
Strt	Word	Start Angle 0 to 360 (0=top)	171
strt	Word	Start select range	185
TeHndl	Long	TEHandle same as ZBasic's TEHANDLE	183
TIB	Var	Transfer Instruction Block	194
tID	Word	Target SCSI ID	194
Time$	Var$	String Variable to store Time String	189
title	Str	String for Control Title	179
title	Str	Title of Dialog Box	185
title	Str	Window Title	177
TxtPtr	Long	Pointer to 1st character of Text	183
TxtPtr	Long	Pointer to text in memory	165
Type	Long	Example: DEFSTR LONG : T&=CVI("TEXT")	164
Type	Long	Scrap Type Example: 'TEXT' or 'PICT'	192
TypList	Var	Pointer to File Type List	190
v	Word	Vertical Position	174
Value	Word	Current control Value	179
Var	Any Variable	Pointer to Any Variable	162
vCell	var	Cell Variable	195
VCmd	Var	Command Source of Bytes	194
Vert	Word	Size in Pixels Height	179
Vis	Byte	True if Visable	177
Visable	Byte	True if Visable	179
Visable	Byte	Visable if True	185
vNext	byte	Flag for next Vert Cell	195
VPoint	Long	Variable With Point Value in it	174
VRect	Rect	Variable Rect Modified by ToolBox Call	174
Vrect	Rect	View Rectangle (Show text in this Rect)	183
Vrect	Rect	View Rectangle	195
Vscrl	Byte	Has Vertical Scroll Bar Flag	195
Wait	Long	number of ticks (timeout)	194
Wdth	Word	Width (in Words)	175

MACINTOSH APPENDIX 

Where	Word	Where the point is on the screen (Global Coordinates).	177
wID	Word	ID of resource Window	177
Wport	Long	Window Pointer	179
WPtr	Long	Window Pointer ZBasic™: Wptr&=WINDOW(14)	177
Wsto	Long	Area to Store Window record (Nil in Heap)	177
xPos	Word	Left edge of horizontal position	165
xSize	Word	Horizontal size in pixels	165
yPos	Word	Right edge of horizontal position	165
ySize	Word	Vertical size in pixels	165

MACINTOSH APPENDIX

MACINTOSH APPENDIX

INDEX OF TOOLBOX ROUTINES

MACINTOSH APPENDIX

MACINTOSH APPENDIX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

ZBASIC

FEATURES AVAILABLE ON ALL VERSIONS

- Same Graphic and File commands make programs work the same way on all computers.
- Not Copy Protected
- Type RUN to compile
- Create commercial quality, stand-alone applications easily
- No Runtime fees
- CHAIN
- Multi-Line IF and FN
- Indents structures: LONG IF, LONG FN, FOR/NEXT, DO/UNTIL, WHILE/WEND
- Labels
- BCD math to 54 digits of accuracy

MACINTOSH

"BEST NEW PROGRAMMING LANGUAGE OF THE YEAR"
Honorable Mention
Editors' Choice Awards
MacUser, 1/87

Creates Fast, Double-Clickable Applications as Easily as Using a BASIC Interpreter.

"...the fastest BASIC available."
MacTUTOR Magazine, 8/86

ZBasic is an ideal choice... because it's a compiler in interpreter's clothing... an attractive alternative to learning C or Pascal."
MacUser (U.K.) 8/86

"If you are a BASIC programmer with a Macintosh, I can't recommend ZBasic highly enough... add a whole new facet to the way you enjoy your Mac."
Neil Shipiro, **MacUser, 12/86**

Special Macintosh features
- Toolbox/ROM support
- Advanced Graphics
- Complete Event Trapping
- MENU, WINDOW, DIALOG
- Laserwriter support
- Imagewriter support
- LongInteger ±2,147,483,647
- MacinTalk support
- AppleTalk support
- Two Powerful Editors
- HELP window under menu
- Read/Write Paint or PICT
- Create Resources
- Just type RUN to compile
- Uses all available memory
- Memory SEGMENT control
- Up to 240 digits of accuracy
- Incredibly fast Compilation

Compare:	Sieve	Sort
ZBasic	7	23
MSBASIC™ comp.	10	649
MSBASIC inter.	662	3335

Available for Macintosh, Macintosh Plus and Mac XL
(512K required)

MSDOS IBM PC & Compatibles

"ZBasic is a powerful offering... provides the flexibility of Turbo Pascal and the speed of compiled BASIC...Kudos to Zedcor and to all users that make the wise decision to use ZBasic to the fullest."
PC Week, 11/85

"a better better BASIC...with a comprehensive and comprehensible manual."
PC WORLD, 8/86

"Of all the compilers reviewed here, ZBasic was the most fun to use. It has real "personality" and a solid feel..."
PC TECH JOURNAL, 12/86

Compare:	Sieve	Sort
ZBasic 4.0	13.7	14
QuickBASIC™	14.0	25
TurboPascal 2.0	14.0	28
BASICA™	2190	3355

NEW 4.0 features:
- **HERCULES** (and plus) graphics support (720x348)
- **CGA** graphics support
- **EGA** graphics support (ZBasic device-independent graphics coordinates are the same with all graphic types. GET & PUT also supported)
- **640K Memory support:** String and real arrays up to available memory.
- **SELECT CASE**
- **NEW! Full Screen Editor**
- Microsoft Mouse™ support
- Extremely efficient RUNTIME. Smaller than any other BASIC
- Creates **.COM** files
- Improved serial drivers with configurable buffer.

Available for MSDOS and IBM PC compatibles
(requires 128K)

APPLE //e, //c, IIGS][and][+

"Unquestionably an excellent implementation of the language
...comes with one of the best user manuals in the industry..."
inCIDER Magazine, 3/86

"ZBasic is a step up for those who have specific needs which outstrip Applesoft..."
CALL A.P.P.L.E., 2/86

DOS 3.3 & ProDOS features
- Inter-mix Text and Graphics in Hi-Res and Double Hi-Res
- Device independent graphics
- Mouse and Joystick support
- "See" pixels with POINT

Special ProDOS features
- **NEW! Full Screen Editor**
- Includes 64 & 128K versions
- Supports RAM disks to 8 MB
- Access to Machine Language
- 16 colors in double Hi-Res
- Customizable Character Sets
- Creates .SYSTEM files

ProDOS Version
Any Apple with 64K or more

DOS 3.3 Version
Apple //e with 128K, //c or IIgs

Compare:	Sieve	Sort
ZBaSIC DOS 3.3	442	170
Applesoft DOS 3.3	3478	4902
ZBasic ProDOS	368	137
Applesoft ProDOS	3614	976
Micol™BASIC	647	354
Hyper C™ (cc)	417	158

Z80 CP/M-80 KAYPRO, TRS-80

"compilation is so fast that execution is indistinguishable from a BASIC interpreter start-up"
BYTE, 5/86

"fast, generates stand-alone programs, requires only modest amounts of memory, has outstanding compilation speeds... and the price is very attractive."
Computer Language, 12/85

Compare:	Sieve	Sort
ZBasic	24	30
MSBASIC inter.	2160	4945

- **CP/M-80 GENERIC** Works on any CP/M Z80 computer. Specify 5.25" or 8".
- **KAYPRO, 2, 4, 10:** Special graphics version. For Kaypros with graphics only (160x100).
- **TRS-80 Model 1,3** (48K-disk)
- **TRS-80 Model 4** (64K-disk) Supports RS and Micro-Labs Hi-Res Graphics (640x240).

Sieve: Ten iterations from Byte, 1/83. **Sort:** Shell-Metzner sort of 2000, 5 character strings (100 character strings with Macintosh). MSDOS, MSBASIC and QuickBASIC tm Microsoft. Applesoft, Macintosh, IIGS, tm Apple Computer, Turbo Pascal tm Borland. Micol tm Micol. Hyper C tm WSM. IBM PC, BASICA tm of IBM. ZBasic tm Zedcor, Inc.

ZEDCOR