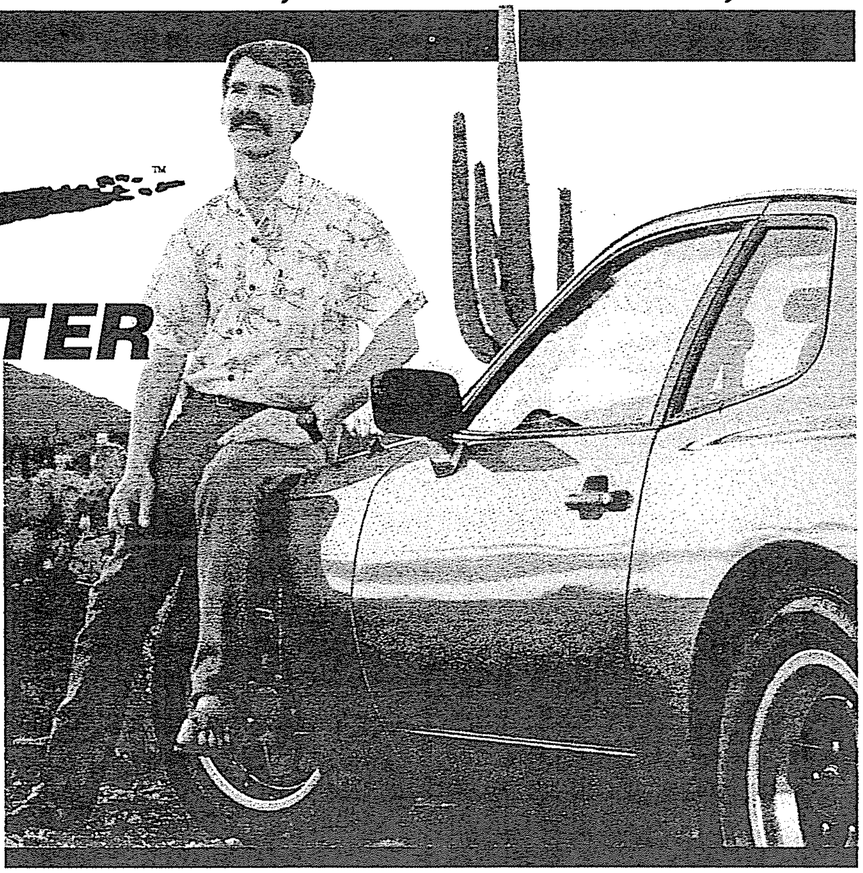
ZBASIC Newsletter **November 15, 1988** **Volume 2, issue 2**

Z™ NEWSLETTER



Andrew Gariepy, President of Zedcor, in a more relaxed pose . This is how he appeared in MacConnection's "President's Catalogue" this August.

ZBASIC 5.0 RELEASED FOR THE APPLE MACINTOSH

In November, ZBASIC 5.0 was released for the Macintosh.

It contains a wealth of new features, including B+Tree file utilities, a new Program Generator that writes source code for you, a toolbox editor, a QuickBASIC™ to ZBASIC conversion program, a manual that is Macintosh specific, and lots more.

The new price of ZBASIC for the Mac is $199.95.

We also have a special offer for ZBASIC owners to buy the new DeskPaint package at a substantial discount.

If you haven't received your upgrade notice yet be sure to contact us. The special upgrade offer expires the end of November.

BETTER LATE THEN NEVER?

I take all the blame for the lateness of this newsletter. When I told you that it would be printed sporadically, I didn't really intend it to be this sporadic.

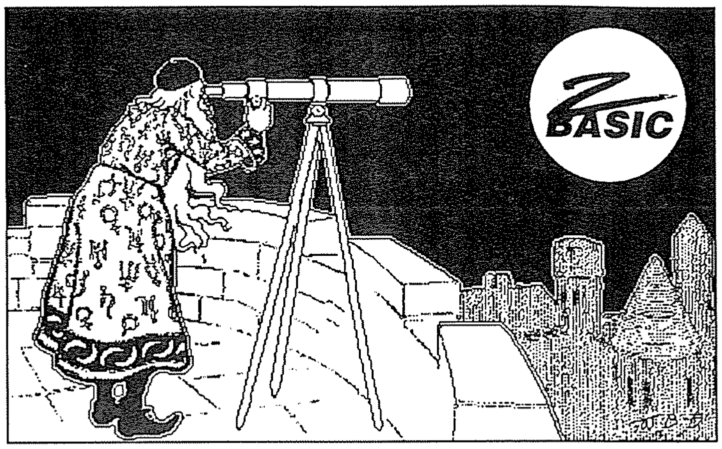
I promise to have another newsletter out in a couple of weeks and the last one out in December.

Ed.

ProDOS ZBASIC 4.21 SHIPPING

ProDOS users with ZBASIC 4.1 or lower should upgrade to version 4.21. It has lots of refinements and has some important bug fixes for serial and chaining statements.

Registered owners call now to upgrade (be sure to have your serial number ready). It sells for $19.95 and comes with two disks (including ProDOS) and a read me file describing the changes.



INSIDE

Z NEWSLETTER

Michael A. Gariepy
Editor in Chief

Andrew R. Gariepy
Technical Editor

ZEDCOR, Inc.

PRESIDENT
Andrew R. Gariepy

CHAIRMAN, CEO
Michael A. Gariepy

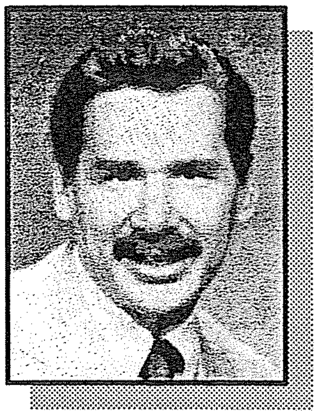
"Z" is published quarterly by Zedcor, Inc., 4500 E. Speedway, Suite 22, Tucson, AZ 85712-5305. (602) 881-8101. Support line: (602) 795-3996.

Please send address changes to the Zedcor address above.

Advertising offices are at 4500 E. Speedway, Suite 22, Tucson, AZ 85712-5305. 1-800-482-4567

"Z" is a technical journal not affiliated in any way with computer manufacturers.

Newsletter created with PageMaker™ 3.0 and Zedcor's new DeskPaint™ 2.0 & DeskDraw™



LETTERS TO THE EDITOR

Dear Editor,

This envelope contains a specific request for help with a programming problem, but I am also taking advantage of my quarter to give you some feedback on features and wishes for ZBASIC 4.01 for the Macintosh. I suspect that this kind of letter gets filed in a different place from programming questions; hence two letters. (*see Dr. Z for question. ed.*)

First let me say that I am generally delighted with ZBASIC 4.01. I am particulary pleased at the easy access to the Macintosh special features like Edit fields, menus, etc. My favorite programming language is pascal, but I am willing to give up sets, local variables, passing variables by reference (LONG FN comes close), etc., in order to be able to include keyboard equivalents in all my menus without having to memorize Inside Macintosh.

I think you have struck a good balance between accessability to rather inexperienced programmers and availability of advanced features.

Still, even a very good language system could be improved, as follows (all of these things are true on the SE and Macintosh PLUS at my laboratory as well as the 512K upgraded to Plus at home):

A. System Crashes:

1. Whenver I compile a program to an application just about the only thing I can do without crashing is to quit ZBASIC. I certainly cannot compile another program the same way-that always leads to a crash. Memory problems or what?

2. In the editor, when I try to erase a blank line by backspacing from the extreme left, I get a crash the next thing I do. It doesn't happen when the cursor is the cross form that indicates the whole line is selected. This crash doesn't happen in any other Editor that I use.

B. Printing Program Listing:

1. The default seems to be Monaco 9 on the first page and Chicago 12 on all subsequent pages. Unless I carefully select Monaco and 9 point from the menus before printing, I get this unpleasant mixture, both at home on the ImageWriter and at the lab on the LaserWrite Plus.

2. I would like to be able to turn off auto indent when listing a program. I indent for clarity on the screen (a great feature of 4.01), but the indentation is then doubled (too much) when printed. how about auto indent on the screen? Could this be a configure option?

C. The Editor:

1. This is a great improvement, and I really appreciate the convenience of the Find feature. There are several ways it could be improved substantially (I have no idea how hard this is to do, and I know that you are primarily interested in writing languages, not word processors but...)

a. Find doesn't find parts of words, words as part of a string in Quotation marks, labels without their Quotation marks, different capitalizations, etc. It would be nice if it worked like the Find in most word processors-giving a choice of whole or part of a word, capitalization, etc. Even I can figure how to program those two, so I am sure you can do it.

b. When a word is found, the Find dialog box obscures half the of the line in which it appears. making searches for a particular use much harder. I would prefer having it find the word in the middle, not the top, of the sceen-context is often very important. (This would also be a better way to have the listing shown when there is an error during compiling.) Others may prefer the line at the top and the dialog box elsewhere-Configure option? But they should NEVER be at the same place.

c. The best I can figure it, Find always starts at the beginning of the program when a new target is being sought. It would be much more helpful if it could start at the current cursor position, thls making it unnecessary to go through dozens of false Finds when a commonly used target is being sought. We can easily put the cursor at the beginning if we want to start there.

d. It would be nice if clicking on the listing window activated it after a find. Most word processors do this, and having to click "Cancel" every time is a pain. Here too, I can guess how to program this, so it must be something that didn't occur to you.

2. In spite of all this "Find" is a vast improvement and really helps. "Replace" would be an even vaster improvement, even more helpful. I know, you're not writing a word processor, but....

D. The Language:

1. I hope that this version will have DIALOG ON included in the POKE PEEK LONG (&904)-&907,0 that was included in the Z newsletter winter edition.

This made it possible for me to use version 4.01 for the first time; I was ready to give up on it after many attempts that I could not understand. Waiting for the newsletter to notify users of that absolutely essential patch (I label the subroutine "Patch 401") was not in the best traditions of Zedcor. An immediate postcard to every user the minute you found out about it would have been better-this was a Major flaw.

2. The CASE command is an important step in the right direction, but it is still a bit cumbersome compared to the Pascal equivalent. If you do more work on it, the ability to handle ranges of values would be very nice, as shown here:

```
SELECT CASE Num%
  CASE 1..5
    GOSUB "Lownumber"
  CASE 6..10
    GOSUB "Mid number"
  CASE 11..20
    GOSUB "High number"
END SELECT
```

Actually I prefer the shorter Pascal version:

```
CASE Num OF
  1..5:   Lownumber;
  6..10:  Mid number;
  11..20: High number;
END;
```

but I suppose it is much too late for such an extensive change.

3. The first time a program is saved after it is opened seems (most of time) to require a confirmation, which I think is not a good

idea- most other programs don't do this. After that, the response to Command-S is a simple save, which is what I would prefer all the time.

4. I don't understand the use of SEGMENT (not SEGMENT RETURN) at all. Is this a way to make certain that ZBASIC breaks the program at safe places-i.e., between subroutines? I can see how SEGMENT RETURN would let the Memory manager purge unused parts of the program (as does "Segment Procedure" in Pascal, but it is not obvious to me just what SEGMENT does by itself. The manual is not too clear on these points; at least I can't understand it.

E. Program Examples:

1. Mostly, I would like more of them. Unfortunately, none of the computer magazines publish any ZBASIC programs, so our only hope of new techniques is from you guys. I would happily pay for a booklet (or disk) of well documented programs, etc. Surely you guys are doing programming that the rest of us will find useful. Even a list of helpful hints would be nice. How about a CompuServe forum for questions and hints?

2. The ZBASIC Construction Set, by and large, is rather disappointing (things cannot be adjusted after they're placed, and I rarely get things right the first time), but possibly source code for that program would give us users a chance to adapt it to our own preferences. What I'd really like is a way to convert MacDraw screens to ZBASIC commands, but this is the real world.

3. It would be a good if keywords that require spaces not be put in example programs (like ENDIF and USR9). Like (I suspect) many others, I configure for "Space Between Keywords" to avoid problems with embedded keywords. This seems to me to the best programming practice, and you guys should set an example by not making this inconvenient.

What has become of the newsletter? I thought it was a good idea and I certainly did not intentionally let my subscription lapse. The last issue I received was the Winter 87/88 issue and we are near the end of summer. If there a problems with the material, I would be glad to contribute some programming hints.

James R. Florini, Ph. D.
Professor of Biochemistry

Dear Dr. Florini,
Thank you for your comprehensive list of suggestions and feedback. We certainly listen to our users and don't get enough feedback from you all.

I guess a lot of you think that writting a letter to us is a waste of time. Let me assure you it is extremly important to use that our customers are satisfied. And we spend a lot of time discussing the ideas and feedback we get. While it is not always possible to answer your letters immediately, we do get back as soon as possible.

I will answer these questions in the order they were given and with reference to the outline numbers.

A.1. System Crashes: This is a tricky question. The problem you described in version 4.01 may have been caused by a couple of things. The most common reason is that running compiled programs that cause instability in memory- those programs that POKE in the wrong place, or programs that access arrays past the last element or assign values to strings greater than their assigned lengths will cause this. By setting array bounds checking and string length checking under the configure options will solve some of these problems.

A.2. The other problem had to do with the Editor. Which has now been stabilized. There were occassions when it would "lose" its cursor reference in memory after a compile and subsequently cause problems. The problem you describe was been completely fixed.

B.1. Fixed in version 5.0

B.2. We agreed with you. But this is easily accomplished by pasting the program directly into a word proccessor and printing it.

C.1.a. The Find dialog box functions the same as the FIND command in ZBASIC. See FIND in the reference section of the manual to understand how it works. It is quite simple when you see the definitions. We will try to put a "text" switch so that it works like a word processor, but no promises.

C.1.b. I agree with you on this. I am working with Andy to see if we can implement it before releasing 5.0. You'll know when you get the upgrade.

C.1.c. Noted. We'll try.

C.1.d. Noted. We'll try.

D.1. You're right. Version 5.0 has this implemented automatically. You were absolutely right about us being remiss to send out a postcard to all our users. This type of thing will not happen again.

D.2. Your Pascal idea for a range of numbers indicator is interesting. We may implement it in a future release. In the meantime us this the following routine I wrote that accomplishes the same thing. Although I agree with you that it is not as simple as the Pascal approach which is bit easier to read.

```
SELECT CASE
  CASE (Num%>0 AND Num%<6)
    PRINT "Lownumber"
  CASE (Num%>5 AND Num%<11)
    PRINT "Mid number"
  CASE (Num%>10 AND Num%<21)
    PRINT "High number"
END SELECT
```

D.3. Noted and fixed.

D.4. SEGMENT is an important part of the memory management routines used in the Macintosh. We have modified ZBASIC slightly to make it more simple.

This is a simple command that tells ZBASIC to start a new SEGMENT at that position. Segments are limited to maximum of about 28K in ZBASIC.

Version 5.0 of ZBASIC no longer will break a segment automatically. If the program compiles to more than about 28K without encountering a SEGMENT statement, it will return a "SEGMENT required error".

Older versions of ZBASIC would just place a SEGMENT at that point automatically. This, of course, would cause system errors and such if the break occurred in the middle of a loop or CASE structure.

You are right that you must manually place your SEGMENT statements.

We recommend that SEGMENT be placed at the end of logical subroutines where it cannot cause problems inside loops and so forth.

SEGMENT RETURN is identical to the regular RETURN statement except that that segment becomes "purgable".

E.1. You'll like version 5.0. It has more examples and they're more structured for easier reading.

E.2. The new "Program Generator" in version 5.0

was created just for that reason. It completely replaces the Construction set and gives you complete freedom to move things around after you place them. You can even reload the files and edit them at a latter date if you want.

It looks so much like MacDraw that you will be pleasantly surprised.

E.3. Noted and fixed in both the new manual and in the example programs. Sorry about that.

The newsletter is in your hands. Sorry for being late. You'll get the next one a few weeks after you receive this one.

Dear Editor,

I am overjoyed to see the ZBASIC 5.0 upgrade [notice]. I have been using 4.01 regularly, but have been extremely frustrated with the editor which crashes my system at times.

For lack of any other toolbox reference with your language, I have begun reading selected articles from MacTutor and it has improved my programming tremendously.

Unless you get someone to write a comprehensive Advanced book (with examples of how to utilize the toolbox) your language will not compete like it should with Microsoft. My God, there must be more than five books out helping Microsoft BASIC users while all we have is a better language and only a reference book. When can we expect a book such as this?

I have recently written my own database program using edit fields and INDEX$. However, I am not exactly sure what is the most efficient way to set up a file. If you could give me your thoughts on how you would save an INDEX$ array, it would be of great help to me. This INDEX$ array has a maximum of 1000 elements with 80 characters possible in each INDEX$. A quick FOR/NEXT loop preceded by an OPEN statement would do me fine.

The way I am doing it now, the file size is way to large to comparable text files. Surely, I am doing something wrong.

I am glad to see DeskPaint doing so well. It will give ZBASIC more exposure I think. More exposure will lead to more books on the subject. I am starved for great reading material. I'm not the only one.

To have extensive examples of how to use RMAKER and ResEdit to develop full fledged ZBASIC applications would make my day (year?). For instance, how to go about creating modal dialog resources, bit map animation, and other toolbox functions.

I am sure many of the questions I have concerning the toolbox with ZBASIC will be answered with the new version!

You have done an excellent job supporting the ZBASIC community (other than the lack of books) and I want to congratulate you on your efforts. You seem to take a customer's opinions more seriously than many other software developers and I think that's great. If your program generator does what the Construction Set was supposed to do and more, then I'm sure that's another welcome addition to the environment. ZMOVER/ZTREE/ZCONVERT also sound interesting.

In closing I'd like to ask you for information regarding writing an article in your "Z" newsletters. I am the chief designed and developer of the CavernQuest™ adventure program due out in 1989. It supports full color animation on the Mac II, object manipulation and digitized sounds and scanned images throughout, and much more. I'll be sure to plug ZBASIC in the credits! By the way, I have many thoughts on large scale development in ZBASIC that I'd like to share with others thorugh your newsletter. Keep up the good work.

Brian Booker
President
ADC Games
P.O.BOX 32360
Columbus, OH 43232

Dear Brian,

Thanks for your comments. We know of two people working on ZBASIC books for the Mac. Be patient. The one for Macintosh will be great.

To save and read an INDEX$ array to disk in the most efficient way, use the following simple routines:

As far as articles for the newsletter, please call me direct. We are always looking for new material. We even pay for good stuff.

Please contact me direct (Mike Gariepy) at 800-482-4567. Your feedback would be very welcome by our readers.

Good luck on your new application. I look forward to seeing it.

```
END                  :' These are subroutines to
                     :' be called from your program.

"Write INDEX$"
'
'   This routines Writes an INDEX$ array to disk. To load it
'   back in use the "Read INDEX$" routine below.

File$=FILES$(0, "Save file as...","Index.DAT",vol%)
IF File$ = "" THEN END   ' Don't load if nothing chosen

OPEN"O",1,File$,,vol%
WRITE#1, LastindexNo&       'How many elements

FOR Count= 0 TO LastIndexNo&: 'Put last element here
  Temp$=INDEX$(Count)       : 'Get the next element
  Length$=CHR$(LEN(Temp$))  : 'Get the length of string
  WRITE#1, Length$;1        : 'Write the length byte first
  WRITE#1, Temp$;ASC(Length$): 'Save the string
NEXT Count

CLOSE#1
RETURN



"Read INDEX$"
'
'   This routines reads an INDEX$ array saved with the
'   "Write INDEX$" routine above.

File$=FILES$(1, "",,vol%)
IF File$ = "" THEN END     : 'Don't load if nothing chosen

OPEN"I",1,File$,,vol%
READ#1, LastIndexNo&       : 'Number of elements in INDEX$

FOR Count=0 TO LastIndexNo&
  READ#1, Length$;1
  Length%=ASC(Length$)     : 'Get the length byte

  READ#1, Temp$;Length%    : 'Load only the characters saved
  INDEX$(Count)=Temp$      : 'Load into INDEX$
NEXT

CLOSE#1
RETURN
```

Dear Editor,

I recently purchased your Z BASIC for the Macintosh SE. Since then, I have enjoyed its versatility and speed. I have been programming in BASIC for a number of yearsl and I have become very good at it. Now that I have adjusted to programming in the Macintosh environment, I would like to begin writing some "Real Macintosh Programs".

Unfortunately, I have discovered some problems which I hope you can help me with.

- I would like to include "snd" resources in my programs. How?

How?

- I would like to create a Desk Accessory with Z-BASIC. How?

- I would like to put small icons on the menu bar. How?

- I would like to include a "True Font Menu" in my programs.

I own ResEdit, and I know how to operate it(although a few of the resources are unfamiliar to me.) I am quite experienced with programming, computers, and Macintoshes. Please answer as many of the above questions as possible. Thank you very much for your attention,

Aaron Segal
13735 Sprucewood Circle
Dallas, TX 75240

Dear Aaron,

In order:

- See the "Playsound.BAS" on the example disk. It allows both synchronous and asynchronous playing of sound. You may play most .snd resource files direct. Or you can use many of the public domain programs like Sound Convertor or the many Hypercard stacks that convert Hypercard .snd resources to other sound type files.

- To add icons to your menus use the caret (^) and the number 1, 2, etc. for icon 257, 258, etc.

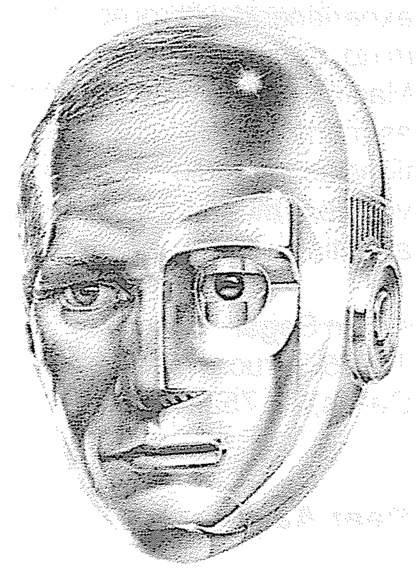
As far as putting icons on the menu bar, you will need to access the toolbox directly and add them. This gets complicated and you are on your own.

- To add a FONT menu to your programs use the new "Program Generator". It has facilities to do this easily.

The new 5.0 manual should be much easier for you to read. I worked hard to make it is easy to use as possible.

DEAR DR. Z

I would be grateful for advise on a (I hope) relatively simple problem that I have encountered in trying to connect a Macintosh SE to a 96-well microtiter plate reader in laboratory. The reader is a device that measures light absorbed by each of the 96 wells in a plastic plate, and it is very useful for a lot of laboratory things. Until now, I have been reading the output using an Apple Pascal program to process the data, and now want to convert to ZBASIC on the Mac so I can make the interface more intelligible to the technicians. I first started using this simple little program (Fig. 1):



```
CLS
INPUT "Press "Return" when ready.";N$
OPEN"C",-1,2400,2,1,1
PRINT "Modem Opened"
MENU OFF
MOUSE OFF
CALL HIDECURSOR
BREAK OFF
"READ"
  READ #-1,A$;0
  PRINT A$;
GOTO READ
```

Here's the nasty culprit that slows this program down.

Fig. 1

I turned off the dialogs to get maximal speed, but the screen still shows a lot of digits are not in the output. They aren't consistant-not every third one or something systematic like that. Indeed, I can't find a specific pattern that would account for what is being lost. As you can see, the baud rate is only 2400, so I can't believe that the program can't keep up. The other settings are the same as I use in my Apple Pascal program so I'm reasonably certain they are correct.

I know from extensive previous experience with this reader that reading each plate transmits 720 bytes; in Apple Pascal I used "Unitread (2, CH,720)", where CH was the array of characters into which I read data. Does ZBASIC have a similar low level FAST way of reading the modem port? I couldn't identify and such command in the 4.0 book but I don't understand all - or most- of the fancy things there. Might I find a way to do this in the dreaded (tremble) Inside Macintosh?

Sincerely yours,
James R. Florin1, Ph.D.
Professor of Biochemistry

DEAR DR. FLORINI,

ZBASIC provides many ways to remedy the problems you described.

The best way to start is to maximize the speed of operations in the "READ" loop by removing the PRINT statement. Since you aren't using the WIDTH LPRINT-2 statement, which significantly improves the speed of printing on the Mac screen, you are really putting a lot of delay in the loop. A simple fix would be to fill up a string before printing it..

The following lines show you how do to this easily.

```
"READ"
READ #-1,A$;0
LONG IF LEN(A$)
  Ch$=Ch$+A$
  LONG IF LEN(Ch$)>250
    PRINT Ch$
    Ch$=""
  END IF
END IF
GOTO READ
```

Another alternative to your loop would be to set a large buffer for the data coming in. This way you will not lose any data coming in unless you set the buffer so small that it cannot hold the excess.

Since you can set a buffer up to 32K it would be unlikely that you would lost characters even with the less efficient PRINT statement used in your program. You can check the status of the buffer with the LOF (-1) statement now included in ZBASIC 5.0.

The ZBASIC equivalent to the Apple Pascal function would be:

```
READ#-1, A$(0);250
READ#-1, A$(1);250
READ$-1, A$(2);220
```

This would load 720 characterted into the first three elements of the array A$.

The only drawback of this is that you would not be able to "Break out" of the program until that many characters was received. A way around that would be to check that the buffer held that many characters first.:

```
"READ"
  WHILE LOF(-1)<720
    TRON X
  WEND
  READ#-1, A$(0);250
  READ#-1, A$(1);250
  READ$-1,A$ (2);220
  TRON X
GOTO "READ"
```

The TRON X will allow you to break out in case something goes wrong.

There is also the off chance that the port settings are incorrect. See the manual to be sure the settings are correct.

DEAR DR. Z

I have the PC and ProDOS versions of ZBASIC and would like to be able to load directories into string arrays. This is important so my programs will look professional.

Lost in Arkansas

Dear Lost in Arkansas,

You're in luck. See the great programs submitted by Greg Branche in this issue under the MSDOS and Apple II sections.

That's all for this issue. Keep those inquiries comming!

■

$ Request for Articles $

The "Z" newsletter is open to submissions for articles pertaining to ZBasic. This includes articles about:

- ❑ Subroutines
- ❑ Graphics examples
- ❑ Cartoons
- ❑ File handling
- ❑ Games
- ❑ Programming Ideas
- ❑ Editor routines
- ❑ Fiction/Non Fiction
- ❑ Math puzzles
- ❑ Educational programs

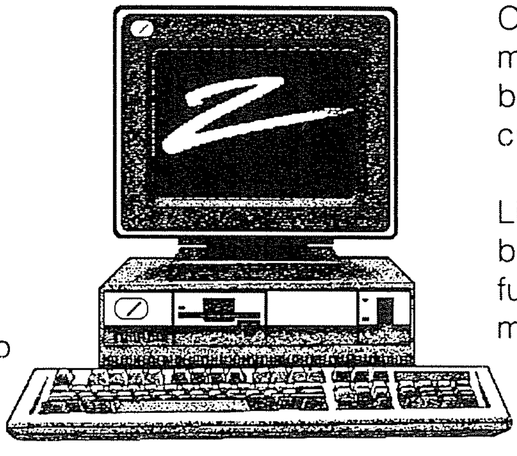
We'll pay $20 to $100 for each article that is accepted (based on size and value). Program submissions will remain in the public domain for all to use.

Articles and Program submissions must be text files (ASCII) on a diskette formatted for IBM PC (360K), Apple II ProDOS or Macintosh.

BEFORE SENDING ANYTHING call Mike Gariepy at 800-482-4567 to discuss submission. This is done to ascertain the desirability of the submission and the amount to be paid.

IBM PC

Reading Directories by Greg Branche



One day a ZBasic user asked me to write a couple of routines for him that would allow his programs to read a disk directory and return the filenames as ZBasic strings. He needed the routines for both the Apple and MSDOS versions of ZBasic. He wanted the routines to return the filenames one at a time so that he could compare the filenames with one that had been entered by the user. That is exactly what these programs do.

The first one I'll discuss is the one written to be used with the MSDOS version of ZBasic. Things were relatively easy here, since MSDOS provides built-in functions to search a disk directory. All you have to do is supply a filename (wildcards allowed) that you wish to search for, and MSDOS will return the first one in the directory that matches your specification. Let's take a look at the program to see how it's done. (Line numbers in the program listings are for reference only. They are not referenced in any way from within the programs.)

Line 00001 simply allocates memory for a simple integer variable and an integer array. The memory contained in the array will be used by the program as a couple of disk buffers for use by MSDOS. Line 00002 simply initializes a couple of pointers to the two buffers within the integer array.

Lines 00006 through 00045 contain the first of two long functions. GetName$ function must be called first, and is used to give the file specification to MSDOS and to retrieve the first matching filename. The filename that is given to MSDOS must consist of eleven characters, no more, no less. This is made up by 8 characters worth of filename, and 3 characters worth of extension (the "xxx" after the period). If you pass a filename that is less than eleven characters to the function, the function will automatically add "?" characters to the end of the filename. This "?" character is the wildcard character.

One other thing to keep in mind is that the directory to be searched will be the currently logged directory.

Lines 14-43 are the assembly language portion of the function. I like to break my machine language lines out into their assembly lan guage equivalents (with the assembly language as comments on the line) to make it easier to see just what is going on. It's also much easier to insert or delete assembly language lines this way.

When the function exits, it returns a string back to the calling program. This string contains the first matching file (if one is found), or a null string (if no match was found).

The second function, NextName$ in lines 49-68, is called to retrieve the second and subsequent filename matches from a directory. Since we've already given MSDOS the file specification by using the GetName$ function, we don't need to pass NextName$ any parameters. Like GetName$, NextName$ returns a string containing the next filename, or a null string if no match was found.

The remainder of the program simply demonstrates how these two functions are used.

```
DIM X, BUF%(127)
dta% = VARPTR(X) + 2 : fcb% = VARPTR(X) + 66
'
'————————————————————————————————
'
LONG FN GetName$(F$)
  ' F$ contains filespec to search directory for
  X = INSTR(1,F$,".") 'extension separated from filename?
  IF X THEN F$ = LEFT$(F$,X-1) + MID$(F$,X+1) 'yes, delete the "."
  'if not a full filename, then pad with wildcard characters
  IF LEN(F$) < 11 THEN F$ = F$ + STRING$(11-LEN(F$),"?")
  StrSeg% = MEM STR 'determine string segment address
  A$ = "" 'initialize A$
  MACHLG &8B,&16,dta%        ' mov  dx,[dta%]         ;address of DTA
  MACHLG &B4,&1A             ' mov  ah,1Ah            ;tell DOS where it is
  MACHLG &CD,&21             ' int  21h
  MACHLG &8B,&3E,fcb%        ' mov  di,[fcb%]         ;address of internal buffer
  MACHLG &32,&C0             ' xor  al,al             ;clear register
  MACHLG &AA                 ' stosb                  ;clear first byte of fcb
  MACHLG &A1,StrSeg%         ' mov  ax,[StrSeg%]      ;get string segment address
  MACHLG &BE,F$              ' mov  si,offset f$      ;get address of file spec string
  MACHLG &46                 ' inc  si                ;point past length byte
  MACHLG &B9,&0B,&00         ' mov  cx,11             ;string is 11 bytes long
  MACHLG &1E                 ' push ds                ;save current segment
  MACHLG &8E,&D8             ' mov  ds,ax             ;point to string segment
  MACHLG &F3,&A4             ' repz movsb             ;move the filespec to FCB
  MACHLG &1F                 ' pop  ds                ;restore segment
  MACHLG &8B,&16,fcb%        ' mov  dx,[fcb%]         ;address of FCB
  MACHLG &B4,&11             ' mov  ah,11h            ;find first filename match
  MACHLG &CD,&21             ' int  21h
  MACHLG &0A,&C0             ' or   al,al             ;was there a match?
  MACHLG &75,&18             ' jnz  nofile            ;no, just exit with a null string
  MACHLG &B8,&0B,&00         ' mov  ax,11             ;else copy the full filename
  MACHLG &89,&C1             ' mov  cx,ax             ;  to the return string
  MACHLG &8B,&1E,StrSeg%     ' mov  bx,[StrSeg%]
  MACHLG &8B,&36,dta%        ' mov  si,[dta%]
  MACHLG &46                 ' inc  si
  MACHLG &BF,A$              ' mov  di,offset A$
  MACHLG &06                 ' push es
  MACHLG &8E,&C3             ' mov  es,bx
  MACHLG &AA                 ' stosb                  ;set string length
  MACHLG &F3,&A4             ' repz movsb
  MACHLG &07                 ' pop  es
  '                  nofile    equ  $
END FN = A$
'
'————————————————————————————————
'
LONG FN NextName$
  A$ = "" 'init A$ to null again
  MACHLG &8B,&16,fcb%        ' mov  dx,[fcb%]         ;address of internal buffer
  MACHLG &B4,&12             ' mov  ah,12h            ;find next filename match
  MACHLG &CD,&21             ' int  21h
  MACHLG &0A,&C0             ' or   al,al             ;was there a match?
  MACHLG &75,&18             ' jnz  nofile            ;no, just exit with a null string
  MACHLG &B8,&0B,&00         ' mov  ax,11             ;else copy the full filename
  MACHLG &89,&C1             ' mov  cx,ax             ;  to the return string
  MACHLG &8B,&1E,StrSeg%     ' mov  bx,[StrSeg%]
  MACHLG &8B,&36,dta%        ' mov  si,[dta%]
  MACHLG &46                 ' inc  si
  MACHLG &BF,A$              ' mov  di,offset A$
  MACHLG &06                 ' push es
  MACHLG &8E,&C3             ' mov  es,bx
  MACHLG &AA                 ' stosb                  ;set string length
  MACHLG &F3,&A4             ' repz movsb
  MACHLG &07                 ' pop  es
  '                  nofile    equ  $
END FN = A$
```

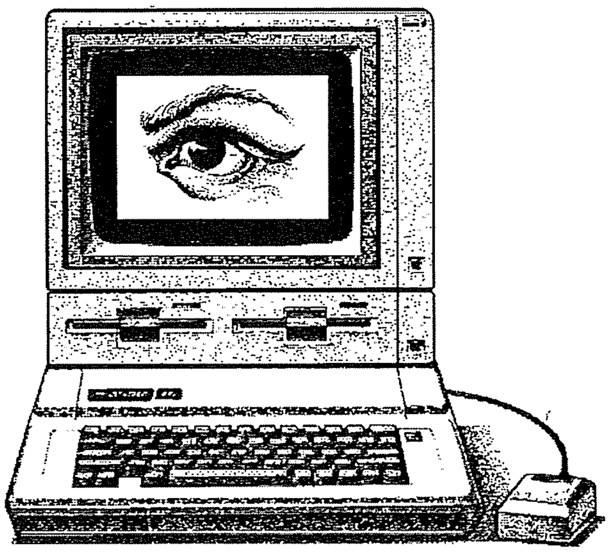
CONTINUED NEXT PAGE...

Continued from previous page.

```
'
'------------------------------------------
'
INPUT “Enter partial filename to find -> ”; Z$
Z$ = FN GetName$(Z$)
LONG IF LEN(Z$)
  Z$ = LEFT$(Z$,8) + “.” + RIGHT$(Z$,3)
  PRINT “LOCATED FILE => ”; Z$
  INPUT “IS THIS THE FILE YOU WANTED? ”; ANS$
  ANS$ = LEFT$(ANS$,1)
  WHILE ANS$ <> “Y” AND LEN(Z$)
    Z$ = FN NextName$
    LONG IF LEN(Z$)
      Z$ = LEFT$(Z$,8) + “.” + RIGHT$(Z$,3)
      PRINT “LOCATED FILE => ”; Z$
      INPUT “IS THIS THE FILE YOU WANTED? ”; ANS$
      ANS$ = LEFT$(ANS$,1)
    END IF
  WEND
  LONG IF LEN(Z$)THEN PRINT “FILE SELECTED -> ”; Z$ ELSE PRINT “NO SELECTION”
XELSE
  PRINT “FILE NOT FOUND”
END IF
```

APPLE II

Reading Directories
By Greg Branche



The routine for the ProDOS operating system on the Apple is a little more complicated. Since ProDOS does not provide the built-in directory searching that MSDOS does, we have to do the work ourselves. It's much more efficient to do this from assembly language, so I wrote the subroutines as a BLOADable modules that get's loaded and executed in the hires graphics video buffer ($2000). In addition, it was written to be used with the 128K version, and so requires the use of a 65C02 processor. (In other words, it won't work with the 64K version unless some of the opcodes are changed.)

The first program, MAKE.DIR.OBJ, simply POKEs the machine language subroutines into memory and then BSAVEs the module to a disk file named DIR.OBJ. Lines 1-26 consist of the BSAVE function. To save yourself a little typing, you can start by loading this function from your ZBasic disk, and then adding the rest of the program to it.

Lines 40-60 contain DATA statements that contain the actual machine language module. When typing these lines in, BE SURE THAT THE VALUES ARE CORRECT. The program simply reads each value from the DATA statements one at a time, and places the value into the correct address in memory. The module is then BSAVEd to the file on the disk for use with the actual directory-reading program.

The binary file consists of four main portions. The first portion consists of vectors to the three subroutines that comprise the directory-reader, plus a 16-bit pointer variable. By setting up the vectors and the variable at the beginning of the memory image, their location will not change in the event that the entry point to one of the subroutines happens to move.

The first subroutine is named "INIT". It takes a string specifying the directory to read, opens that directory file, and then initializes some internal variables. The second subroutine, named "READ", is the actual workhorse of the program. It reads the directory and fills a ZBasic string buffer with the next available filename, or NULL if no more filenames are available. The third subroutine, "CLOSE", simply closes the directory file once you are done with it. A generic ZBasic CLOSE statement probably would have done the same thing, but I don't like to take chances.

Now, how do we use this little goodie? That's what the third program listing is all about.

"DIRECTORY.BAS" BLOADs the DIR.OBJ file into memory, and then reads and displays each filename in the directory one at a time and then displays a count of the active files that it encountered. (Nothing really earth-shattering here. Just remember, this is a sample program!)

Lines 1-20 are the BLOAD function that can be found on your ZBasic disk. As in the previous program, you can save yourself a little typing here by loading it first, and then keying in the remainder of the sample program. One more thing worth mentioning (and I should have mentioned it for BSAVE too) is that the addresses in lines 2, 6, 12, 17, and 19 (12, 14, 16, 22, and 25 for BSAVE) must be adjusted for use in the 128K version. The two functions supplied on disk have been set up for use with the 64K version. Make sure these addresses match those in the listing printed here.

Line 22 loads the machine language module into memory. Line 23 initializes some ZBasic variables to make it easier to use the various subroutines. The entry points for the subroutines are:

```
INIT - $2000
READ - $2003
CLOSE - $2006
POINTER - $2009
```

By setting these addresses into integer variables, it makes it much easier to access these addresses through the use of names that actually mean something.

Line 25 allows the user to enter a string containing the name of the directory to read. Line 26 pokes the address of that string into the module's pointer, so that the module knows which file to open. Line 27 calls the INIT subroutine to open the directory and get the variables initialized. If there is no problem during the initialization, the POINTER variable will be returned with a value of 0. If there are problems, the ProDOS MLI error number will be returned instead. Lines 28 and 29 check for, and handle, any error.

Line 30 pokes the address of a ZBasic string into POINTER. This string must be at least 16 bytes long (the maximum length of a ProDOS filename, plus the length byte). The READ subroutine is then called in line 31. The subroutine will scan the directory until it discovers the first active filename entry, and then will copy the filename into the ZBasic string provided. If no more active files remain in the directory, the length byte of the string will be set to 0 (in other words, a NULL string will be returned).

After reading and displaying all active filenames in the directory, the program calls the TERMINATE subroutine to close the directory and clean up it's internal variables, and then displays the count of the number of active files encountered.

As they stand now, these programs don't have all the bells and whistles that they could have. With a little modification, the methods presented here could add a great amount of professionalism and user friendliness to your programs. If you have any comments or suggestions, I can be reached by electronic mail on GEnie at [G.BRANCHE1], or on AppleLink-Personal Edition at [G Branche]. Have fun! ■

```
00001 LONG FN BSAVE(Path$, Fnum%, Adrs%, Length%)
00002   REM First we have to create the file
00003   POKE &1F00, 7 : REM 7 Parms for create call
00004   POKE WORD &1F01, VARPTR(Path$): REM Set pathname pointer
00005   POKE &1F03, &C3 : REM ALLOW FULL ACCESS
00006   POKE &1F04, 6 : REM MAKE IT A BIN TYPE FILE
00007   POKE WORD &1F05, Adrs% : REM Set AUX_TYPE field
00008   POKE &1F07, 1 : REM Seedling storage type
00009   FOR x = &1F08 TO &1F0B
00010     POKE x, 0 : REM Zero out creation date and time
00011   NEXT
00012   MACHLG &A9, &C0, &20, &0865 : REM Create the file
00013   REM Now we can open the file we just created
00014   POKE WORD &1F03, &AC00 - (Fnum% * &400): REM Point to file buffer
00015   POKE &1F00,3 : REM 3 parms for open
00016   MACHLG &A9, &C8, &20, &0865 : REM Go open the file
00017   REM Now that it's open, write the bytes
00018   POKE &1F01, PEEK(&1F05) : REM Move reference number
00019   POKE &1F00, 4 : REM 4 parms for write
00020   POKE WORD &1F02, Adrs%
00021   POKE WORD &1F04, Length%
00022   MACHLG &A9, &CB, &20, &0865 : REM Write the bytes
00023   POKE &1F00, 1
00024   REM Make sure the file is closed!
00025   MACHLG &A9, &CC, &20, &0865
00026 "End Bsave" END FN
00027 :
00028 CLS : PRINT "POKING DIR.OBJ INTO MEMORY" : PRINT
00029 PRINT "ADDRESS -> "
00030 FOR I = &2000 TO &2142
00031   READ A
00032   PRINT @(11,2) I
00033   POKE I,A
00034 NEXT I
00035 PRINT : PRINT "WRITING FILE TO DISK"
00036 FN BSAVE("DIR.OBJ",1,&2000,&143)
00037 PRINT : PRINT "DONE!"
00038 END
00039 :
00040 DATA &4C,&0B,&20,&4C,&AC,&20,&4C,&37,&21,&00,&00,&AD,&09,&20,&8D,&01
00041 DATA &28,&AD,&0A,&20,&8D,&02,&28,&A9,&0A,&8D,&00,&28,&20,&00,&BF,&C4
00042 DATA &00,&28,&90,&07,&8D,&09,&20,&9C,&0A,&20,&60,&AD,&04,&28,&C9,&0F
00043 DATA &F0,&04,&A9,&4A,&80,&EE,&A9,&03,&8D,&00,&28,&A9,&00,&8D,&03,&28
00044 DATA &A9,&22,&8D,&04,&28,&20,&00,&BF,&C8,&00,&28,&B0,&D7,&A9,&04,&8D
00045 DATA &00,&28,&AD,&05,&28,&8D,&01,&28,&A9,&00,&8D,&02,&28,&A9,&26,&8D
00046 DATA &03,&28,&9C,&04,&28,&A9,&02,&8D,&05,&28,&20,&00,&BF,&CA,&00,&28
00047 DATA &B0,&B2,&AD,&23,&26,&8D,&12,&28,&AD,&24,&26,&8D,&13,&28,&AD,&25
00048 DATA &26,&8D,&14,&28,&AD,&26,&26,&8D,&15,&28,&9C,&19,&28,&9C,&1A,&28
00049 DATA &A9,&02,&8D,&18,&28,&A9,&04,&18,&6D,&12,&28,&8D,&16,&28,&A9,&26
00050 DATA &69,&00,&8D,&17,&28,&9C,&09,&20,&9C,&0A,&20,&60,&AD,&09,&20,&85
00051 DATA &64,&AD,&0A,&20,&85,&65,&A9,&00,&92,&64,&AD,&16,&28,&85,&62,&AD
00052 DATA &17,&28,&85,&63,&38,&AD,&19,&28,&ED,&14,&28,&AD,&1A,&28,&ED,&15
00053 DATA &28,&B0,&59,&B2,&62,&F0,&0F,&29,&0F,&92,&64,&A8,&B1,&62,&91,&64
00054 DATA &88,&D0,&F9,&EE,&19,&28,&EE,&18,&28,&AD,&13,&28,&CD,&18,&28,&90
00055 DATA &10,&18,&A5,&62,&6D,&12,&28,&85,&62,&A5,&63,&69,&00,&85,&63,&80
00056 DATA &27,&20,&00,&BF,&CA,&00,&28,&90,&12,&C9,&4C,&D0,&1F,&AD,&14,&28
00057 DATA &8D,&19,&28,&AD,&15,&28,&8D,&1A,&28,&80,&11,&A9,&01,&8D,&18,&28
00058 DATA &A9,&04,&85,&62,&A9,&26,&85,&63,&B2,&64,&F0,&98,&A5,&62,&8D,&16
00059 DATA &28,&A5,&63,&8D,&17,&28,&60,&A9,&01,&8D,&00,&28,&20,&00,&BF,&CC
00060 DATA &00,&28,&60
```

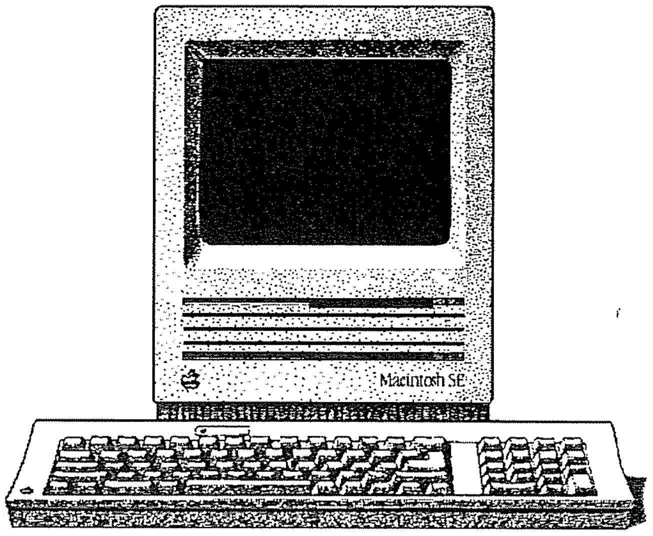
```
LONG FN BLOAD (File$,Fnum%,Adrs%,Length%)
  Buffer% = &AC00 - (Fnum% * &400)
  POKE WORD &1F01, VARPTR(File$) : REM Set up parmlist
  LONG IF Adrs% = 0 : REM Use Address in directory
    POKE &1F00, 10 : REM 10 parms for this call
    MACHLG &A9, &C4, &20, &0865 : REM Get the file info
    Adrs% = PEEK WORD (&1F05)
  END IF
  IF Length% = 0 THEN Length% = &FFFF
  POKE WORD &1F03, Buffer%
  POKE &1F00,3
  MACHLG &A9, &C8, &20, &0865 : REM Open the file
  POKE &1F01, PEEK(&1F05) : REM Get ProDOS Reference number
  POKE &1F00, 4: REM 4 parms for read
  POKE WORD &1F02, Adrs%
  POKE WORD &1F04, Length%
  MACHLG &A9, &CA, &20, &0865 : REM Read the file into memory
  POKE &1F00, 1
  MACHLG &A9, &CC, &20, &0865 : REM Make sure the file is closed
“End Bload” END FN
:
FN BLOAD(“DIR.OBJ”,1,0,0) : REM BLOAD the machine code
Init = &2000 : Read = &2003 : Terminate = &2006 : StrPtr = &2009
:
INPUT “Please enter the directory name -> ”; F$
POKE WORD StrPtr, VARPTR(F$) : REM Point to directory pathname
CALL Init : REM Open the directory and init variables
Err = PEEK WORD (StrPtr) : REM Check for any errors
IF Err <> 0 THEN PRINT “ProDOS error $”; HEX$(Err) : STOP
POKE WORD StrPtr, VARPTR(A$) : REM Point to ZBasic string
CALL Read : REM Read the first filename
WHILE LEN(A$)
  PRINT A$
  count = count + 1
  POKE WORD StrPtr, VARPTR(A$)
  CALL Read : REM Read the next filename
WEND
CALL Terminate : REM Must close the directory when we're done!
PRINT count; “files are in the directory.”
END
```

MACINTOSH

If you haven't already received your upgrade for ZBASIC 5.0 be sure to do so soon.

It is easily the finest BASIC compiler ever made for any computer!

Perhaps the neatest thing added to ZBASIC 5.0 is the "The Program Generator". It is a completely new program for generating ZBASIC source graphically. It is as close as you can come to real MacDraw-like program.



This program was written by Chris Stasny, a loyal ZBASIC programmer that wasn't happy with the ZBASIC Construction set. He said he could write a program generator that was much more powerful. I told him to go for it. Six months later he plopped the program on my desk and I must say I was very impressed.

So impressed in fact, that I wanted to make sure ALL ZBASIC users had it.

In addition to the Program Generator we have included the ZMover Toolbox editor, also by Chris Stasny. It allows you add, delete and modify toolbox routines and even create your own custom machine language commands.

The ZTREE B+Tree file utilites are incredibly useful as well. With them you can create your own giant data bases with little effort.

They are 30K of ZBASIC source code in the form of LONG FN's. You simply add the source code to your program and use one simple function call to add, delete or find items in your index instantly. Multiple indices may be used so you can sort your data by any number of fields.

And, best of all, the new manual is Mac-specific. It is really a joy to use and will save you lots of time.

I'm still looking for articles for the Mac section of the newsletter. I have some great submissions from Frank Turovich this month.

Thanks Frank.

As for the rest of you... Get on the stick!

ed.

Another way of doing Hierarchical Menus

by L. Frank Turovich

With introduction of the Mac II several new features were added to the managers that resided in the Mac's ROM. One of these, the Menu Manager introduced hierarchical menus.

(There is another great example of doing hierarchical menus on the new ZBASIC 5.0 disk in the Program examples folder. ed.)

To set a series of program choices with the old Menu Manager, required the program to display a modal dialog box and force the user to choose from an array of button options. Hierarchical menus, however, allow the user to quickly select from a menu any of many options without a dialog box. A hierarchical menu is a sub-menu that appears next to a regular menu item and offers a subset of item choices. In Figure #1 we see a sample where the Print option offers three print qualities in a submenu instead of a dialog box.

To create hierarchical menus we will use ResEdit to create our menus resources and use several Tool-box calls to implement them. If you do not have ResEdit, see page E-32 in the ZBasic manual for instructions on creating a resource file with RMaker which is included on your ZBasic disk.

First, start up ResEdit and select New to create a resource file for our program. We'll name this file "Menu.r". Double-click on Menu.r to open it's window. Select New to open the TYPE's dialog. Scroll until you find the resource type "MENU". Select it and click in the OK button to create your resource. ResEdit automatically opens a MENU window. Choose New again to create the Print menu.

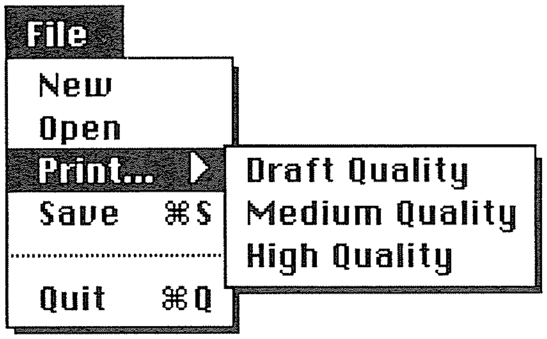
Type in the following information:

MenuID = 250.
enableflgs= $0000000F

Ignore the size boxes but enter the menu's title Print.

To add an item to the Print menu, select the five asterisks under the Print title and choose New. ResEdit will create a menu item. Follow the example in Figure #1 to complete the Print menu by adding all three items. Close the Print menu when done.

Using the same technique, create the File menu using Figure #1 as a guide.



MenuID =100
enableflgs =$0000005F.

Close the File menu when done.

Now is the time to enter the special information to link our two menus together. Select the File menu and use the Open General menu option. If you copied the example your window should look like Figure #2.

Search the data until you find the Print item. The next two blocks of data are what we are interested in.

The hex code 1B is what the menu manager uses to decide if there is a hierarchical menu for that particular item. The hex code FA (250 decimal) is the submenu's MenuID. Since our Print menu has a MenuID of 250, that will be the submenu. Click next to the area to change and enter the 1B and the FA. Close the File menu.

One last thing and we are done. When ResEdit created the menus it gave each of them a unique ID number. We need to change that to reflect our chosen MenuID numbers. Use the Get Info option to select each file and change the ID field to the MenuID number. Finally, close and save the Menu.r file.

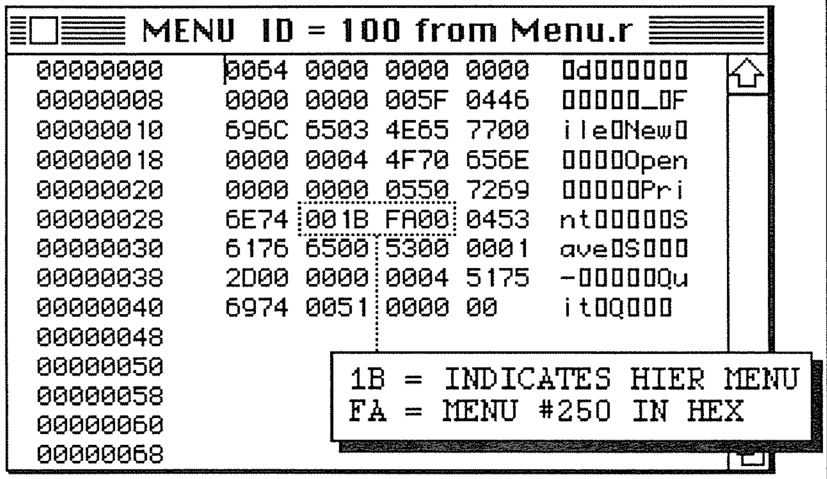
We have created our hierarchical menus, now let's use them.

To use our resource menus requires ToolBox calls, specifically GETMENU & INSERTMENU. GetMenu retrieves menus identified by our MenuID number. InsertMenu writes the menu to the menu list in the order specified by the Before parameter. If Before is zero, the menu is written to the menu list. If Before is -1, the menu is inserted next to the item identified by the 1B designator, in this case, the Print item. Finally, we use the CALL DRAWMENUBAR to display our menu list to the screen.

Hierarchical menus can be nested five levels deep. In other words, a submenu can have a submenu, can...well, you get the picture. Also, only 255 submenus are allowed. To have multiple submenus, simply change each MenuID into its hexadecimal equivalent and enter 1B and each MenuID in hex next to each item where the submenu will appear.

That's all there is to it. Use the program 'HierMenu.BAS' to see your hierarchical menus in action.

continued next page...


MENU ID = 100 from Menu.r

```
00000000   0064 0000 0000 0000   Od0000000
00000008   0000 0000 005F 0446   00000_0F
00000010   696C 6503 4E65 7700   ile0New0
00000018   0000 0004 4F70 656E   00000pen
00000020   0000 0000 0550 7269   00000Pri
00000028   6E74 001B FA00 0453   nt00000S
00000030   6176 6500 5300 0001   ave0S000
00000038   2D00 0000 0004 5175   -00000Qu
00000040   6974 0051 0000 00     it0Q000
00000048
00000050
00000058
00000060
00000068
```

1B = INDICATES HIER MENU
FA = MENU #250 IN HEX

Figure #1

FIGURE #2

```
' - ResEdit Menus in ZBasic

Ref%=FN OPENRESFILE("Menu.r")           ' - OPEN RESOURCE FILE
FileM%=100 : PrintM%=250                ' - DEFINE MENU ID'S

FileMenu& = FN GETMENU(FileM%)          ' - GET FILE MENU
PrintMenu& = FN GETMENU(PrintM%)        ' - GET PRINT MENU
CALL INSERTMENU(FileMenu&,0)            ' - INSERT FILE MENU TO MENU LIST
CALL INSERTMENU(PrintMenu&,-1)          ' - INSERT PRINT MENU TO MENU LIST
CALL DRAWMENUBAR                        ' - DRAW MENU LIST TO SCREEN

ON MENU GOSUB "MenuEvent"
MENU ON                                 ' - ENABLE MENU EVENTS
DO                                      ' - MAIN LOOP
UNTIL Done
MENU OFF                                ' - DISABLE MENU EVENTS

"MenuEvent"
MenuID=MENU(0) : ItemID=MENU(1)         ' - GET MENU NUM & ITEM NUM
PRINT "You chose Item #";ItemID;" of Menu #";MenuID;", the ";
SELECT CASE MenuID                      ' - USE CASE FOR EASY READING
       CASE 100 :                       ' - GET FILE ITEMS
       SELECT CASE ItemID  ' - GET ITEM & RESPOND
              CASE 1 : PRINT "New Item"
              CASE 2 : PRINT "Open Item"
              CASE 3 : PRINT "Close Item"
              CASE 4 : PRINT "Save Item"
              CASE 6 : GOTO "QuitProgam"
       END SELECT
       CASE 250 :                       ' - GET PRINT ITEMS
       SELECT CASE ItemID
              CASE 1 : PRINT "Draft Quality"
              CASE 2 : PRINT "Medium Quality"
              CASE 3 : PRINT "High Quality"
       END SELECT
END SELECT
MENU                                    ' - UNHILITE MENU
RETURN                           ' - RETURN MAINLOOP

"QuitProgam"
CALL CLOSERESFILE(Ref%)                 ' - CLOSE RESOURCE FILE
END                                     ' - PRIOR TO ENDING
```

CREATING DIALOGS IN RESEDIT

by L. Frank Turovich

Open your copy of ResEdit and wait for it to load. We are going to use one of ResEdits most powerful menu commands, New from under the File menu. With this command we can create any resource our program requires.
Before we can start we need a file to store our resources in. Select New from the File menu and ResEdit requests a name for our new resource file. Let's call it 'Dialog.r'. Type that into the edit field and press OK. ResEdit creates 'Dialog.r' and opens up its resource window.

Select New again and this time a dialog appears with a list of the resource types ResEdit can create. Scroll until you see the four letter resource type DLOG and double-click on it or type DLOG into the available field and press OK. ResEdit creates a resource of type DLOG. Open the resource DLOG by double-clicking. Select New again. Is this getting familiar now? ReSEdit now creates a DLOG resource with a unique ID number. Make a note of the ID number as we will need it later.

Click open the new DLOG resource to look at the default window ResEdit creates. To position the window, point, click and drag. To adjust the size, click near the lower right corner and drag until it appears correct in the miniature screen display.

Now, we need to set the attributes of our dialog window. Go to the DLOG menu and select 'View by Text'. We now see the dialog window with all its attributes displayed. See Figure #1 for field information.

Select 'Display Graphis' from the DLOG menu to view your new dialog. Double-click on our miniature dialog and the full size version appears. Now we can fill it with items such as buttons, edit fields and icons.

Select New and ResEdit automatically creates an item for your dialog. Since the push-button item is already selected, we'll start with it. Type OK into the edit field and click the windows close box. Now, use the mouse to position and size the pushbutton where you want it. Do the same for the other items. Create a static text item, an icon item, and an edit field item. Remember their respective item numbers for our program later. See Figure #2 for possible dialog display.

The last thing to do is set our dialogs ID number. Select 'Get Info' and change the resource number displayed to 1000, our dialogs new ID number. Close the window and your new ID numbered DLOG is displayed. Now close all windows in 'Dialog.r' and save at the alert prompt.

That's it. You've now created a unique dialog for your program.

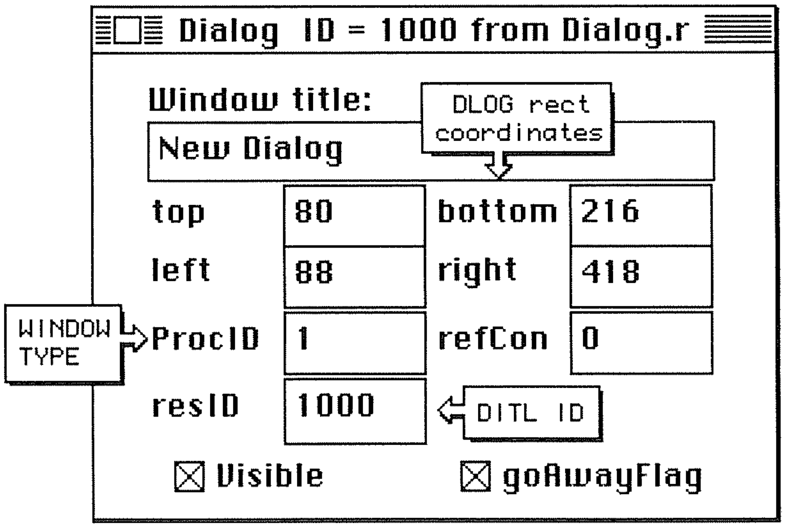
The routines in the demo program 'Dialog.BAS' can now be used to call and control your dialog box. They will call your dialog and display it, accept input to the edit field, then retrieve that input and write it to the screen.

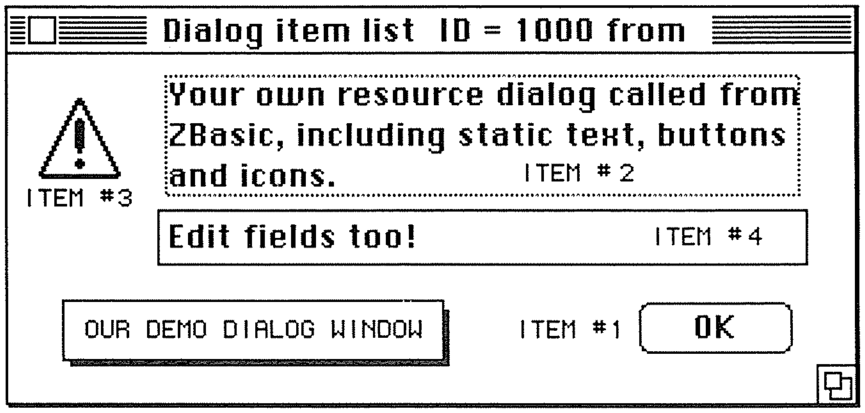
One final warning, do not mix normal ZBasic windows with toolbox windows as this will cause several system errors.

continued next page...

continued from previous page...

Title	: If you want one
Left,Top,Right,Bottom	: Dialog screens coordinates
ProcID	: Window types (See E-156 for info)
resID	: ID number of DITL used
refCon	: Reference value
Visible	: Dialog shown when called
GoAwayFlag	: Close box in window




Dialog item list ID = 1000 from
Your own resource dialog called from ZBasic, including static text, buttons and icons.
ITEM #2
ITEM #3
Edit fields too!
ITEM #4
OUR DEMO DIALOG WINDOW
ITEM #1
OK

```
REM ZDialog.BAS
REM
REM Demontrates toolbox calls to use dialogs
REM in your ZBasic programs

DIM R%(3)
Ref%=FN OPENRESFILE("Dialog.r")          : 'Get our resource file
Err%=FN RESERROR                         : 'Check for file error
LONG IF Err%<>0                          : 'If yes, quit
BEEP : GOTO "QuitProgam"
END IF

"SampleRoutine"
CLS : PRINT "ResEdit Dialog Sample..."
DlgID%=1000 : GOSUB "GetDialog"          : ' Look for our dialog In file
PRINT "Finished"
"QuitProgam"
CALL CLOSERESFILE(Ref%)                  : ' Get Rid Of resource file
END

"GetDialog"
Efld%=4
DlgPtr&=FN GETNEWDIALOG(DlgID%,0,-1):    : ' Get dialog
GOSUB "BtnOutline"                       : ' Outline main button
GOSUB "WaitLoop"                         : ' Wait for button press
CALL GETDITEM(DlgPtr&,Efld%,IType%,IHand&,R%(0))
                                                : ' Get handle to edit field
CALL GETITEXT(IHand&,EFStr$)             : ' Get edit field text
CALL DISPOSDIALOG(DlgPtr&)               : ' Remove dialog from screen
PRINT EFStr$
RETURN

"BtnOutline"
ItemNum%=1                               : ' Points to OK button
CALL GETPORT(OldPort&)                   : ' Save Old Port for Later
CALL SETPORT(DlgPtr&)                    : ' Set Port to open dialog
                                         : ' Get handle to item to be outlined
CALL GETDITEM(DlgPtr&,ItemNum%,IType%,IHand&,R%(0))
LONG IF IHand&<>0                        : ' Got Item Handle TO Use
  CALL PENSIZE(3,3)                      : ' Adjust pen
  CALL INSETRECT(R%(0),-4,-4)            : ' Standard Outline Offset
  CALL FRAMEROUNDRECT(R%(0),16,16)       : ' Draw Outline
END IF
CALL SETPORT(OldPort&)                   : ' Restore Old Port
RETURN

"WaitLoop"
DO
CALL MODALDIALOG(0,ItemHit%)             : ' Wait for ItemHit
UNTIL ItemHit%=1                         : ' If not OK button, Continue
RETURN
:
:
```

REGISTERED OWNER INTRO

by L. Frank Turovich

The wave of copy protection is passing, and behind it software makers are resorting to moral and ethical value systems to protect their products. One of the more popular forms appearing is to have the owner enter his name when he first boots up the new software. Thereafter, the owner's name appears on the opening screen for all to see.

ZBasic makes it easy to include such a routine into your applications.
The following Macintosh program illustrates how to open the data fork of your compiled program, get and save the owners name, then display it each time the program is restarted.

CheckOwner is actually a very simple routine to use. The key, however, is using the DEF OPEN command to reset the applications CREATOR & TYPE when the owner is saved, otherwise the application will become a file.

```
"CheckOwner"

Owner$=""
AplName$="<< Compiled Program Name >>"
OPEN "I",1,AplName$,,Vol%
LONG IF LOF(1)<>0 ' - CHECK FILE FOR OWNER
      INPUT#1,Owner$
END IF
CLOSE#1
WINDOW 1,"IntroWind",(100,50)-(410,150),-2
TEXT 0,12,0
LONG IF Owner$<>""
' - IF THE OWNERS NAME IS PRESENT - DISPLAY IT
      CALL MOVETO(20,25)
      PRINT "This software registered to :"
      CALL MOVETO(20,45) : PRINT Owner$
      DELAY 5000   ' - ALLOW TIME TO READ
XELSE
' - IF THE OWNERS NAME IS EMPTY - GET & SAVE IT
' - THIS SHOULD ONLY OCCUR ON INITIAL STARTUP
      CURSOR 0
      CALL MOVETO(20,25)
      PRINT "Enter your name as owner :"
      EDIT FIELD 1,"",(20,35)-(300,50)
      BUTTON 1,1,"OK",(200,65)-(300,85)
' - LOOP WAITING FOR BUTTON PRESS
      DIALOG ON    ' - ENABLE DIALOG EVENTS
      DO
            Dlg%=DIALOG(0)
      UNTIL Dlg%=1
      DIALOG OFF   ' - DISABLE DIALOG EVENTS
      Owner$=EDIT$(1)     ' - GET USERS NAME
      DEF OPEN "APPL????"      ' - RESET FILE TO APPL
                  ' - ???? = YOUR CREATOR
      OPEN "O",1,AplName$,,Vol%
      PRINT#1,Owner$     ' - SAVE OWNERS NAME
      CLOSE#1      ' - TO DATA FORK
      CURSOR 4
END IF
WINDOW CLOSE 1     ' - CLOSE WINDOW
RETURN
```

ShutDown-Power Routine
by Frank Turkovich

This Macintosh specific command, SHUTDOWN, is mislabeled. Restart would be more appropriate, since that is really what it does *(Old timers will remember that Apple changes Shutdown to both Restart and Shutdown in newer versions of the system. ed).*

The ShutDown statement ejects all disks from the Mac and then reboots your system.

The machine language routine called ShutDown-Power performs a true shutdown just like the Finder. It ejects all disks, shuts down the system and then displays an alert box with the option to reboot.

This may be handy for some of you:

```
"ShutDownPower"
MACHLG &H3F3C,&H0001,&HA895
```

See us a MacWorld Expo in San Francisco! Booth #2210 in the Moscone Center.

ZBASIC NEWSLETTER

"Z" Newsletter
ZEDCOR
4500 E. Speedway, Suite 22
Tucson, AZ 85712-5305

Bulk Rate
U.S. Postage
PAID
Tucson, AZ
Permit number
2220

Send to: