

A Review of AC/BASIC: A BASIC Compiler For The IIGS In A Desktop

Ken Kashmarek

Prior to the announcement of the IIGS, program development was fragmented across the Apple II computer line, with DOS 3.3, Apple Pascal, CP/M, and ProDOS. With the IIGS and ProDOS16, the door has finally been opened for a uniform operating system environment, supporting all languages. To date, there are several assemblers, a Forth compiler, a C compiler, at least two Pascal compilers, and now three BASIC compilers, all operating under ProDOS 16. This month, we will take a look at an innovative BASIC compiler for the Apple II world.

After the announcement of GSBASIC by Apple Computer, Inc., several new products have come to the market which support BASIC on the IIGS. While others support Applesoft and/or GSBASIC, the AC/BASIC compiler is noteworthy because it supports the Microsoft BASIC language. (See Microsoft sidebar.)

Notice the term "interpreter." An interpreter is one which "compiles" a statement every time it is executed. It requires several machine language instructions to process each BASIC element, e.g., PRINT. A compiler analyzes the BASIC program and "converts" it into machine language statements for each statement. The result is stored in a separate file resulting in a faster operating program because each statement has been converted to machine language form and fewer instructions are executed compared to the interpreter process.

AC/BASIC offers IEEE and decimal math, dynamic arrays, the desktop environment, support for super high resolution graphics, and sound generation using the sound chip. The compiler is derived from Absoft's 32 bit BASIC compiler, marketed by Microsoft as the MS BASIC compiler for the Macintosh.

Microsoft

"Applesoft was one of the earliest BASIC interpreters developed by Microsoft. Microsoft was founded by Bill Gates and Paul Allen after they sold BASIC to MITS for the Altair microcomputer."

Cecil Fretwell

A Powerful Alternative

The strength of the AC/BASIC compiler is the Microsoft BASIC language. Existing Macintosh MS BASIC programs can be ported to the IIGS with few changes. AC/BASIC also opens

the door for applications from other computers which use MS BASIC.

Based on significant experience from QuickDraw on the Macintosh, the compiler supports QuickDraw II on the IIGS using PALETTE, CIRCLE, LINE, and COLOR commands, as well as MENU, WINDOW, BUTTON, and DIALOG commands. IIGS toolbox knowledge is not required to access the most popular graphic features. This allows fast development of sophisticated applications without using the QuickDraw II documentation. For example, the GSBASIC demo program called PICS is over 250 lines long. The AC/BASIC version of this program is less than 60 lines.

The IIGS sound chip is supported with the SOUND, WAIT, RESUME, and REPEAT commands. Complex instrumentation, multi-timbre sound, and synthesizer sequencing effects can be developed.

Hardware Requirements

AC/BASIC is distributed on one 3.5" disk (IIGS System Disk V3.1), including more than 100 example programs. 512K of memory is required for normal execution, while the AC/BASIC version of the PICS demo program, requires 768K. (See BBS sidebar.)

The compiler takes approximately 45K of memory, and stays resident for compiling and editing. The editor takes less than 30K, and is a version of the editor available with ORCA/M by Byte Works, Inc. The run time library takes 75K and is loaded with the compiled program, or dynamically loaded after the program begins execution. Additional memory is used by the RAM based IIGS toolset routines (80K to 100K). The executing program may use whatever remaining memory is available. (See Run Time sidebar.)

The documentation is over 400 pages and is well written. There is an appendix dealing with IIGS toolbox routines, mostly the QuickDraw II functions. Example programs are used throughout the manual to illustrate compiler features.

BBS

"Absoft has developed a demonstration disk available on many bulletin board systems. At the time of this article's preparation, the demonstration disk is on at least DELPHI, CompuServe, and GENIE. It is also available from Absoft."

Cecil Fretwell

Run Time

"The use of a run time library in conjunction with compiled code is common practice even on larger computer systems. A program is compiled to produce a machine language module which contains subroutine calls to modules in the run time library. For example, there are subroutines to add, subtract, multiply, analyze decisions, etc. Run time modules can often be included with the compiled code resulting in a larger file, or the run time library can be loaded once and shared by one or more programs contained in smaller files. One of the disadvantages of the run time concept is that the same code is always loaded resulting in some modules which are never used by a program. Obviously, this eats up memory."

Cecil Fretwell

22 Call -A.P.P.L.E.

Features

Some of the more important features include:

- Dialect of Microsoft BASIC language.
- IEEE or decimal math (32 and 64 bit floating point).
- Dynamic arrays and strings.
- Independent subprograms.
- Labeled statements.
- Block IF...THEN constructs.
- FOR...NEXT constructs.
- WHILE...WEND loops.
- Global and local variables.
- IIGS sound capabilities.
- Super high resolution capabilities.
- INCLUDE source files.
- Batch compilations.
- Sequential and random access files.
- Generated code is position independent and reentrant.
(See *Position Independent/Reentrant sidebar.*)
- Desktop applications and event handling.

Position Independent/Reentrant

"A vast majority of machine language code using assembly language requires the code to always reside in a specific area of memory. This is called position dependent code. To allow the code to reside in another area requires a re-assembly or special user code to relocate the code.

Position independent code may be loaded into any area of memory. This is done without making any changes to the code. Producing position independent code is not an easy task.

Producing reentrant code is more difficult. Normally associated with subroutines, it literally means a subroutine can call itself or a program/subroutine can be used by other programs 'at the same time.' This means the reentrant code must always save values of registers, variables, etc., unique to each call, and restore these values when it exits."

Cecil Fretwell

Noted Differences

At this point in the life of the IIGS, many developers of new applications are familiar with the APW or ORCA/M programming environment. This means using an assembler or compiler followed by a separate link process with the advanced linker, all of which runs from the command oriented shell interface. AC/BASIC does not require a shell program, nor does it use a separate linker process. (See Linker sidebar.)

AC/BASIC is launched directly from the Launcher or Finder and establishes the desktop environment for compiling (see Figure 1 for the compiler options dialog). It does not provide a command or file handling interface. The compiler translates source code directly into position independent machine code. AC/BASIC performs its own internal linking to the run time libraries. Object code compatibility is not supported for other languages, so only the AC/BASIC run time library is needed for execution.

The run time library can be internally linked with the compiled program as part of the load file, or be dynamically loaded at program execution time. The price for elimination of the advanced linker process, is the loading of the complete 75K run time library at execution time. When the load time is combined

Linker

"Technically speaking, a compiler creates a machine language file called an object module. A linker is a mechanism to combine one or more object modules into what is called a program or load module. An object module may refer to code not included within its space. A linker resolves this problem by linking the code to the proper code in other modules.

The concept of linking can result in a considerable reduction in program development time. Including all modules in a single source file increases the compile time for a program. Compiling separate modules to remove syntax errors, etc., then linking all the modules together, saves time and leads to good habits in the form of modular or structured programming."

Cecil Fretwell

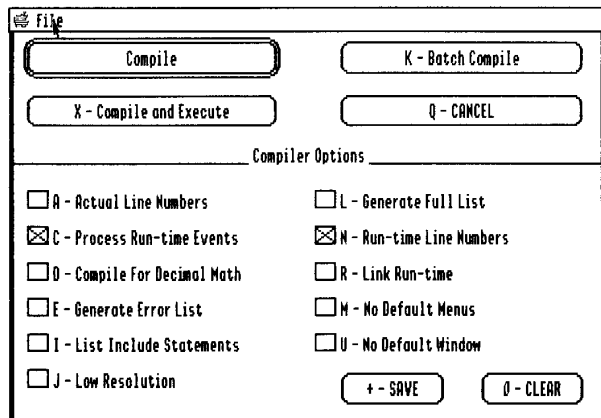


Figure 1

with initialization of the RAM based toolbox routines, it takes just under 10 seconds to start up a program from a RAM disk, more than 20 seconds to start up a program from a hard disk drive (depending on file locations), and 30 seconds from a 3.5" drive. These times can be somewhat improved by running the configuration program supplied with the compiler.

Unlike a command oriented shell, the default environment for the compiler and the compiled program is the IIGS desktop, using QuickDraw II and super high resolution graphics. However, AC/BASIC does not use an editor running on the desktop. It reverts to a text editor. The editor is suitable and gets the job done, however, it seems a bit out of place in the super high resolution desktop environment of the compiler.

AC/BASIC Language

If you have ever used Microsoft BASIC on any other computer, you will be right at home with AC/BASIC. The manual is familiar territory for experienced users. For Applesoft and GSBASIC users, it will take a short time to become familiar with the more common language features.

With an interpreter like Applesoft or GSBASIC, it is important to use short variable names to improve execution speed. (Applesoft will allow more than two characters in a name, however, they will be ignored.) This often generates programs which are hard to read or understand. With a compiler, variable names can be longer and more descriptive. AC/BASIC supports

May 1988 23

variable names up to 40 characters long, beginning with a letter, and consisting of letters, digits, or periods. Numeric constants are converted to executable form at compile time. In particular, constant expressions are converted at compile time for faster execution at run time.

Variable types are:

Variable Name	Type	Range
A\$	string	0 to 32767 characters
A%	integer	-32768 to 32767
A!	single	1.18E-38 up to 3.4E38
A#	double	2.23D-308 up to 1.79D308

Define type statements, DEFSTR, DEFINT, DEFNG, and DEFDBL, may be used to establish the definition of a variable based on the first letter of the variable name.

Arrays are dynamically defined at execution time. The OPTION BASE statement is used to define the lower bound (0 or 1) for array subscripts. ERASE is used to delete an array definition.

Strings include the use of DATE\$, TIME\$, INSTR, and MID\$ for selecting characters from a string, or replacing characters in a string, and many others.

Line numbers may be used in programs, although labels are preferred. A line number or label is not required for each line. The editor does not support the RENUMBER command, so line numbers would be extremely difficult to deal with anyway.

While GOSUB and RETURN may be used for subroutines, the high level SUB...END SUB construct is a much better way of organizing a program. SUB...END SUB provides parameter passing to the subroutine, plus local variables within the subroutine.

Structured programs may be built using the IF...ELSE...ENDIF constructs and WHILE...WEND loops. ELSE IF may also be used with the IF...ENDIF construct. These capabilities do not exist in Applesoft and only in a weak form in GSBASIC.

File Handling

Some file statements are different. CHDIR (change directory) is used to set the prefix, while KILL is used to delete a file. NAME...AS is used for renaming a file, and FILES is the same as a CATALOG command. The FILE\$ function is used to invoke the standard file selector from the desktop (like the IIGS program launcher screen).

Non-disk file names are "SCRN:" for screen output, "KYBD:" for keyboard input, "LPT1:" for printer output, "COM1:" for a serial I/O port, and "CLIP:" for the clipboard. The print manager is always invoked when printing listings or information generated by a program. The use of the device "LPT1:PROMPT" displays the style dialog box. (See Notation sidebar.)

Without directly coding a toolbox call, the user can immediately take advantage of desktop interfaces for file and device selection by using AC/BASIC standard features (FILE\$ and LPT1:PROMPT):

```
10 OPEN "LPT1:PROMPT" FOR OUTPUT AS #2
20 OPEN FILE$(1,"TXTSRC") FOR INPUT AS #1
30 WHILE NOT EOF(1)
40   LINE INPUT #1,A$
50   PRINT #2,A$
60   PRINT A$
```

```
70 WEND
80 CLOSE
```

See Figures 2, 3, and 4 for the dialogs presented by the toolbox routines for printer and file selection. Figure 4 is the same as what you will see when opening a file for compiling or editing.

For Applesoft programmers, file handling in AC/BASIC is new and different. It will take some time to master the new capabilities. These capabilities have been in use for several years on other computers. Most of the remaining file commands are similar to the GSBASIC file commands.

Graphics

AC/BASIC supports the super high resolution environment directly with high level language capabilities. PRINT puts characters directly on the same screen with super Hi-Res graphics and colors, using 320x200 mode or 640x200 mode.

SCREEN 1 sets 320x200 while SCREEN 2 sets 640x200. WINDOW creates or displays a window with given attributes such as size, location, scroll bars, etc. Multiple windows may be on the same screen. Windows may be opened, closed, or switched. Output to a window is text (using PRINT) or graphics using pixel commands, or LINE, CIRCLE, and SCROLL.

PALETTE and COLOR are used to control super high resolution palettes and colors. LCOPY is used to dump a screen to a printer including color.

AC/BASIC supports array based graphics for handling rectangular images. GET and PUT move images from and to the screen. PICTURE based graphics allows output of drawing commands to a string to be used later for display on the screen. The PICTURE strings may be stored in arrays or files.

To support the desktop environment, AC/BASIC provides a full set of event handling commands, including MOUSE, DIALOG, BUTTON, EDIT FIELD, and MENU. There are also TIMER, BREAK event, and ERROR handling capabilities.

Full desktop capabilities can be used without access to the IIGS toolbox documentation or knowledge of the toolbox interfaces used by AC/BASIC.

Sound

The SOUND statement is used for sound production. WAVE is used in conjunction with SOUND to further control the sound production. With AC/BASIC commands, the sound chip can be used to produce noises, special effects, musical notes, or entire songs. Notes are controlled through the use of the time interval and duration parameters. They overlap if the interval is less than the duration. Tones range from 0 to 127 where middle C is tone #60.

The WAVE statement is used to tailor sounds. This includes setting up different instruments and wave forms such as SIN, SQR, TRI for sine, square, and triangle waves.

Full control of the power of the IIGS sound chip may require

Notation

"For one accustomed to Applesoft and the world of BASIC.SYSTEM or DOS 3.3, commands like KILL and terms such as 'LPT1' seem strange. They come from the world of Microsoft BASIC. A form of this same BASIC is provided on systems painted blue, therefore, to some the notation is quite familiar."

Cecil Fretwell

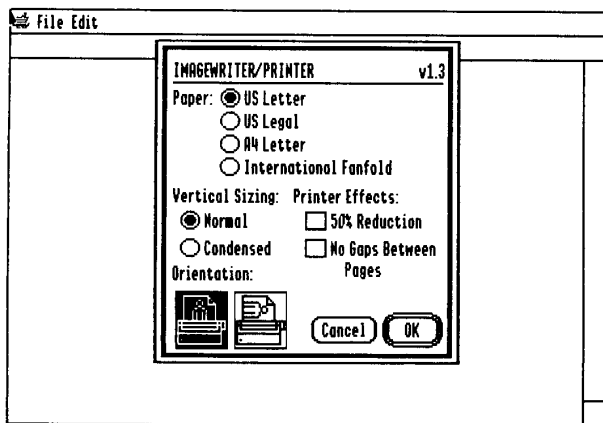


Figure 2

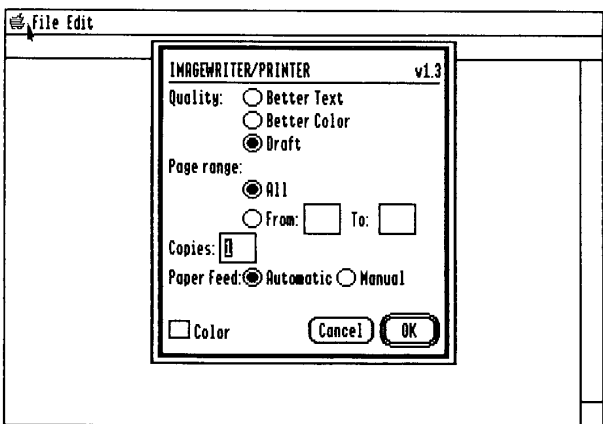


Figure 3

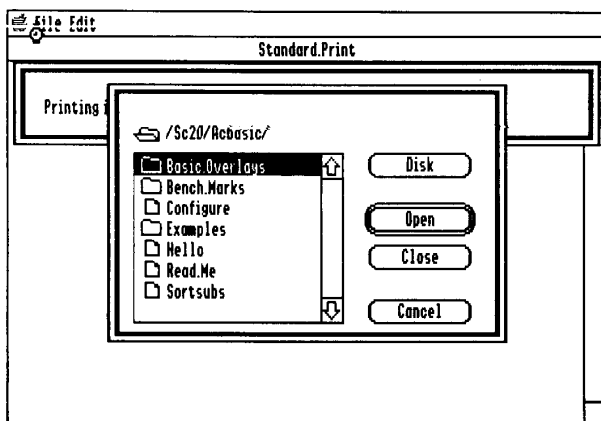


Figure 4

use of the toolbox reference. However, AC/BASIC provides sufficient power to use this feature immediately. The sample HELLO program (Listing 1) provides a complete sound demo.

Looks Good — Sounds Great...

AC/BASIC is a significant advance in use of the BASIC language on the IIGS. It is not, however, without problems.

While the compiler was derived from the 32 bit compiler for the Macintosh, Absoft chose not to implement 32 bit integers. Using 32 bit integers would be faster than using floating point operations where only the extra precision of long integers is needed.

The desktop is the default environment for the Macintosh. Not so on the IIGS. For example, the FINDER and LAUNCHER turn off super Hi-Res by reverting to text mode when starting a new program. The new program must restart the super Hi-Res environment. This takes time. The screen is cleared and the RAM based tools are reloaded. Since the IIGS does not provide the capability for the tools to be left active and shared by many applications, tool shutdown and restart is the normal mode of operation.

AC/BASIC is handicapped by this limitation. A complete cycle of compiler startup, source file editing, program compilation, and execution, will see at least nine screen changes. More if you count each screen change when tools are shutdown. The time penalty is generally 10 to 20 seconds from a 3.5" disk. Of course, this is related to AC/BASIC using and generating a default desktop environment.

AC/BASIC cannot generate anything but a desktop application. In other words, no text screen processing is allowed. If you want 40/80 column text screen applications or command driven programs, then this compiler is not for you. This can be important since text handling on the super Hi-Res screen is slower than text on the standard 80 column text screen. The desktop operation is dependent on the IIGS tools, therefore, the compiler is not at fault.

There is no support for other languages or external run time libraries of user written routines. Assembly language programs are supported via the MACHINE and BLOAD statements. However, this capability does not use the system loader and does not support relocation of symbols. The assembly language program must be position independent. This is a serious handicap considering the extensive capabilities provided by the IIGS for program loading. However, it is offset somewhat by the fact that AC/BASIC is very powerful and should not require use of assembly language programs.

The compiler must always be loaded for each compile and run sequence. The compiler does not make use of the restart from memory option provided by the system loader. This is not a big problem since the compiler is small. The editor, like the compiler, is also loaded from disk each time it is used. It should also use the memory restart option. The real problem is loading 75K of run time modules every time a program is executed (loaded dynamically or with the program load file). This, combined with the startup of RAM based tools, takes 30 seconds from a 3.5" drive. An option is available to speed up loading which bypasses the Font Manager initialization, but with some restrictions.

The manual describes the search path for the run time libraries BASIC.1L and BASIC.2L:

1. Current folder.
2. Current folder's BASIC.OVERLAYS folder.
3. Root folder of boot disk.
4. SYSTEM folder of boot disk's root folder.

Appendix H describes another location as the BASIC.OVERLAYS folder in the SYSTEM folder (recommended for hard disk installation). While dynamic loading of the

run time library cuts down on file size, it is offset by a lot of disk searching. If you boot from a hard disk, you cannot launch the compiler from a different disk. It will not find the run time libraries on the other disk. The run time libraries must be on your boot disk and in the folder with the source files.

The compiler and editor do not take advantage of IIGS memory clipboard. The editor cut/paste operation goes to a disk file. The editor always saves a source file to disk and the compiler must always read the file from disk (it could be compiled from memory). The compiler may create and delete three or four work files on disk for each compile (in the current source file directory). This degrades compiler performance, and leaves a copy of the work files in each source file directory. There are no options for directing the work files to a RAM disk for faster execution. For speed, you should copy your source file to a RAM disk before using the compiler.

Overall, the entire edit, compile, and execution of an AC/BASIC program is not smooth. When the compile and run option is selected, and program execution is complete, return always goes to the Launcher (or Finder). This is annoying when you want to immediately make changes for another test. However, I expect this, and other desktop transition difficulties to improve over the life of the product.

But The Speed...

When dealing with compilers, significant emphasis is placed on speed of the object code. In many cases, speed comparisons are useful and sometimes they are even valid. Absoft provided a set of programs and produced a speed comparison table. After looking at the table and running the programs, my observations indicate the compiler produces code significantly faster than Applesoft and GSBASIC for numeric calculations, but not much faster for string operations or desktop operation.

If you rely on the speed comparison provided by the vendor, be sure you examine the programs and understand what the comparison is providing. The vendor included the loop overhead in the timings. For small, simple programs, the load times offset any performance improvements provided by compiled code. The best comparison which can be made is to compile one of your programs and check the results. This is the improvement you should expect. Be certain you are using integer, single precision, and double precision where appropriate in your program, and that calculations have been optimized before and after the comparison.

In particular, speed comparisons are a result of the code generated and the function being compared. In the case of AC/BASIC, the floating point numeric calculations are significantly faster than the IIGS SANE package used by other vendors (AC/BASIC does not use SANE but provides its own implementation of IEEE math which is as much as 3:1 faster than SANE for single precision and 2:1 faster for double precision).

The Vendor Response

I have used a beta release of the compiler and V1.0. The review was written after using V1.1. The compiler was tested on an Apple IIGS with a one megabyte memory expansion card and the level 01 ROM. Several errors I found have been fixed in V1.1. I believe the vendor will be responsive to any problems you find.

Absoft chose to use the IIGS Print Manager tool interface for printing. I could not get the interface to print correctly. It always double spaces output, both in text and better text modes. I even used the control panel settings to remove line feeds after carriage

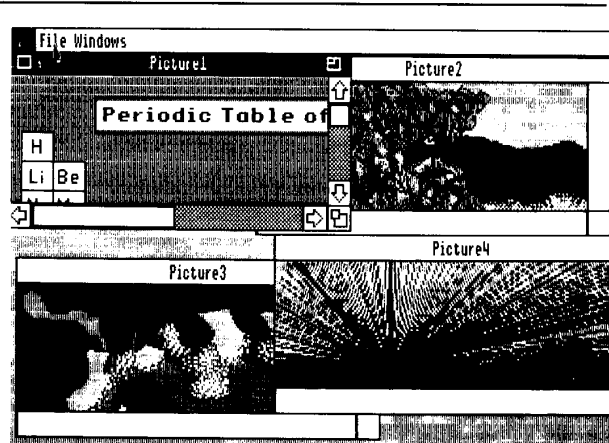


Figure 5

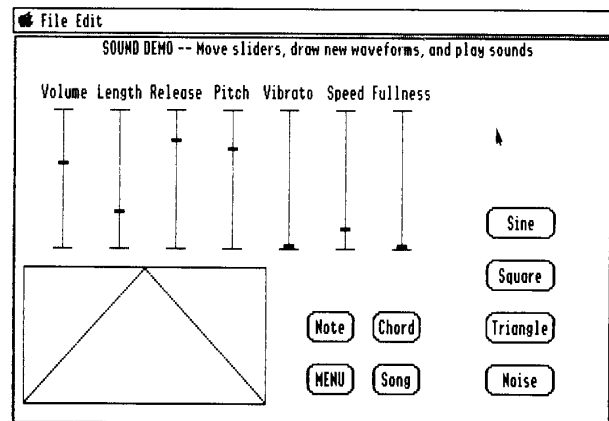


Figure 6

returns, and this made no difference. The same file prints correctly from APW with no printer changes or control panel changes. The documentation did not provide any information about changing printer settings. The weakness is probably in the Print Manager interface. If draft mode printing cannot be supported at ImageWriter speeds, the interface should not be used.

Absoft has stated that they are considering changes to the compiler with regard to the user interface and improvement to the turnaround time for program development. At this point, I believe the vendor has provided a quality product which you should find useful. ■

Please rate this article on the Reader Service Card by circling:

61 Excellent 62 Good 63 Fair 64 Poor

Product:

AC/BASIC Compiler

Company:

Absoft
2781 Bond Street
Auburn Hills, Michigan 48057
(313) 853-0050

Price: \$125.00

CIRCLE 65 ON READER SERVICE CARD

LISTING 1

```

' PICS Demo written in AC/BASIC for the Apple IIGs.
' Reproduced with permission by Absoft.
' This version of PICS differs from other versions by
' sizing the pictures to the entire window rather than
' using scroll bars to view them. PICS should be compiled
' with the A,C, and N options on

SCREEN 2+4
$ABOUT "PICS Demo"
DEFINT A-Z
OPTION BASE 1
DIM Pictures(32768/2,4)

MENU 1,0,1,"File"
MENU 1,1,1,"Quit"

MENU 2,0,1,"Window"

FOR x=1 TO 4
    MENU 2,x,1,"Window"+STR$(x)
    WINDOW x,"Window"+STR$(x),(x*50,x*20+10)-(x*50+319,x*20+109),1
NEXT x

WINDOW 5,"",(220,80)-(420,120),3
PRINT
PRINT "          Loading pictures..."

FOR x=1 TO 4
    OPEN "pic"+MID$(STR$(x),2,1) AS #1
    BLOAD #1,Pictures(5,x),32768-320
    CLOSE #1
    Pictures(1,x)=640
    Pictures(2,x)=188
    Pictures(3,x)=128
    Pictures(4,x)=160
    WINDOW OUTPUT x
    PUT (0,0)-(WINDOW(2),WINDOW(3)),Pictures(1,x),PSET ' Draw picture
NEXT x

ON MENU GOSUB HandleMenu
ON DIALOG GOSUB HandleDialog
MENU ON
DIALOG ON
WINDOW CLOSE 5
WHILE 1 : CONT : WEND

HandleMenu:
    IF MENU(0)=1 THEN END
    WINDOW MENU(1)
    WINDOW OUTPUT MENU(1)
    MENU
    RETURN

HandleDialog:
    d0=DIALOG(0)
    IF d0=3 THEN WINDOW DIALOG(3)
    IF d0=4 THEN WINDOW CLOSE DIALOG(4)
    IF d0=5 THEN
        WINDOW OUTPUT DIALOG(5)
        PUT (0,0)-(WINDOW(2),WINDOW(3)),Pictures(1,DIALOG(5)),PSET
    END IF
    RETURN

```

' Use the 640x200 screen without window
' Set-up text for the "About.." box
' Use integers to speed execution
' Start array elements at 1
' Space for four 32K pictures

' Create a "File" menu
' ...with one "Quit" item.

' Create a "Window" menu

' Create 4 numbered menu items and 4
' windows with titles on a diagonal

' Create another small window
' To display the 'loading' message

' Load and display 4 pictures
' Open picture file
' Skip first 4 elements of array
' Close the picture file
' Set width and height (640x188)

' Set 640x200 mode for each

' Set the current output window

' Draw picture

' Set-up Menu event trapping
' Set-up Dialog event trapping
' Turn on menu trapping...
' ...and dialog trapping
' Close the "Loading..." window
' Infinite loop to wait for events

' Quit was selected
' Bring the selected window to front
' Make it the current output window
' Unhighlight menu bar
' Return to where menu event occurred

' Process automatic dialog events
' Get the type of dialog event
' WINDOW OUTPUT DIALOG(3)
' Window selected
' A Close Box was pressed
' A window needs to be redrawn
' Make it the current output window
' Draw picture
' Return to where dialog event occurred