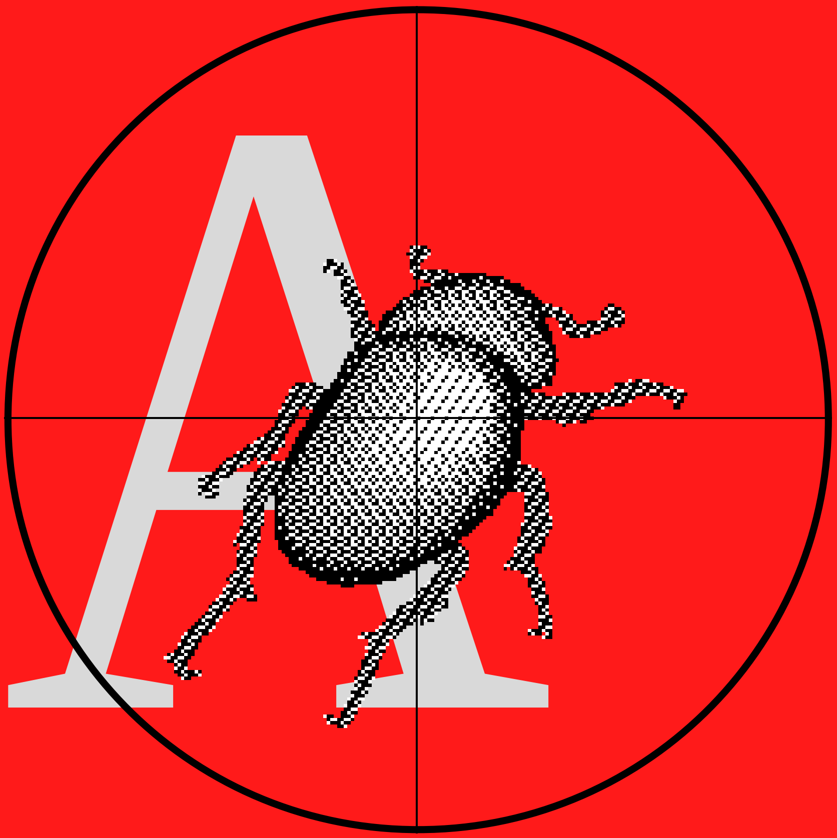
RADE

Real-time Applesoft Debugging Environment



Morgan Davis Group

Copyright

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without prior written permission of the author. No patent liability is assumed with respect to the use of the information contained herein. **WHILE EVERY PRECAUTION HAS BEEN TAKEN IN THE PREPARATION OF THIS PRODUCT, MORGAN DAVIS GROUP ASSUMES NO RESPONSIBILITY FOR ERRORS OR OMISSIONS.**

The product names mentioned in this manual are the trademarks or registered trademarks of their manufacturers.

ProDOS and ProDOS BASIC are copyrighted programs of Apple Computer, Inc. licensed to the Morgan Davis Group to distribute for use only in combination with this product. Apple software shall not be copied onto another diskette (except for archive purposes) or into memory unless as part of execution of this product. When this product has completed execution, Apple software shall not be used by any other program.

Apple Computer, Inc. makes no warranties, either express or implied regarding the enclosed software package, its merchantability or its fitness for any particular purpose. The exclusion of implied warranties is not permitted in some states. The above exclusion may not apply to you. This warranty provides you with specific legal rights. There may be other rights that you may have that vary from state to state.

First printing — October 1992 — U.S.A.

Printed on
recycled paper.

Contents

Introduction

Chapter One: Getting Started

Chapter Two: RADE Commands

Chapter Three: A Debugging Session

Introduction

What Is RADE?

RADE is the Real-time Applesoft Debugging Environment. RADE helps you explore the guts of a BASIC program, its variables, program flow, and other runtime characteristics without disturbing the program's memory or screen display. Without RADE, the debugging process, an experience that accompanies the development of any program, is a frustrating, time consuming, and arduous task. With RADE and its powerful features, like stepping, tracing, and breakpoints, debugging is quick and painless.

Notation

Throughout this manual the following symbols are used to denote keys on your keyboard:

RESET	Reset	DEL	Delete
OPTION	Option	↑	Up arrow
⌘	Open-apple	↓	Down arrow
CTRL	Control	←	Left arrow
ESC	Escape	→	Right arrow
ENTER	Return	↹	Tab
SPACE	Space	SHIFT	Shift

Hyphenated key references, such as ⌘-ESC, tell you to press and hold the first key while typing the second.

This manual displays commands in uppercase, but neither RADE nor BASIC are case sensitive.

Credits

Software	Russell E. Gibson
Manual	Morgan Davis
Publishing	Morgan Davis Group 10079 Nuerto Lane Rancho San Diego CA 91977-7132 USA
Support	+1 619 670 0563 +1 619 670 9643 FAX +1 619 670 5379 BBS
Testers	Amrit Chauhan Jon Thomason Morgan Davis

RADE was initially conceived, designed, and developed during the summer of 1992. Daily electronic mail (via the ProLine network) and phone calls between San Diego (Morgan) and Hillsboro (Russ) kept the project rolling. Russ and Morgan met for the first time, face to face, only after RADE and the manual were completed.

Getting Started

This chapter gives a brief overview of RADE, how it works, and how to install and activate it. Sit in front of your computer while following along.

What You Should Know

Since RADE is a BASIC programmer's utility, knowledge of Applesoft and ProDOS BASIC (BASIC.System) is assumed.

RADE runs only on the Apple IIGS. However, the Applesoft BASIC programs you develop and test with RADE can be used on any Apple II series computer, provided they do not do anything IIGS-specific.

RADE does not modify or change your programs in any way. Simply use RADE to find bugs in a program, then repair them as you normally would.

NOTE: RADE must be installed ***before*** *any other BASIC extensions, or any programs that modify HIMEM.*

How RADE Works

RADE is a binary program (BIN) file that you install in memory by running it from Applesoft BASIC. Once installed, RADE remains quiet and unobtrusive until you need it.

To activate RADE, press ⌘-CTRL-DEL. This sequence tells RADE that you want to begin debugging. When Applesoft attempts to execute the next statement, RADE halts the program and saves the screen display. It then switches to its own display where you can do your debugging.

You can examine the contents of variables, step through a few statements, set some breakpoints, or turn on statement tracing (these features are discussed later). When you're done, RADE restores the program's display and it continues where it left off.

If your program requires changes, you can make them in Applesoft's immediate mode and continue testing. When your program is bug-free, you can remove RADE from memory by unloading it.

Installing RADE

As with all commercial software, you should make a working backup of your RADE disk. Do that now.

If you have a hard disk, copy RADE to the directory where you normally keep your BASIC programming utilities.

To quickly install RADE from the working copy of the RADE disk, insert the disk into a drive, then perform one of the following steps:

- ► Restart the computer to boot the RADE disk, or
- ► Launch the Startup file on the RADE disk

To install RADE from the copy on your hard disk:

- ► Launch BASIC.System (if not already there)
- ► Run RADE using the appropriate pathname:

```
]-/HD/DEV.TOOLS/RADE   ENTER
```

NOTE: RADE uses auxiliary memory, occupied by the /RAM volume. If there are files on /RAM, RADE asks if you really want to install it, thus removing any files stored there. If you must use a RAM disk, the Apple IIGS can set aside memory to support /RAM5, which doesn't conflict with RADE .

When installed successfully, RADE displays:

```
RADE 1.00 Installed
```

Now, work in Applesoft as you normally would. RADE is completely dormant and won't interfere with your programs. It occupies only 768 bytes of main memory, so you may not even notice its impact on free memory.

Activating RADE

RADE is usually activated by pressing ⌘-CTRL-DEL while a program is running. This is the Apple IIGS keyboard flush sequence, used to remove any keystrokes from the built-in keyboard buffer. It also instructs RADE to begin intercepting Applesoft program statements as they're encountered.

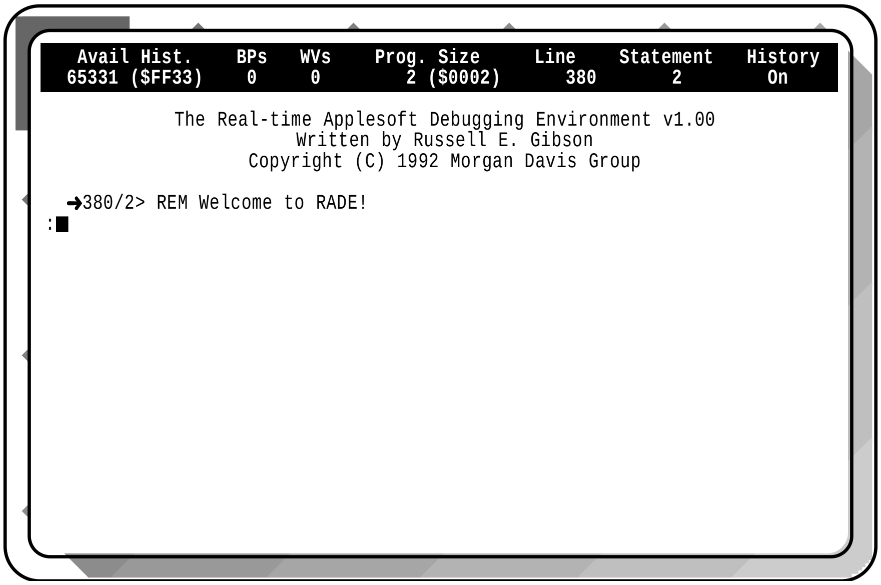
If you press ⌘-CTRL-DEL while in an input mode (e.g., immediate mode or during an INPUT or GET), RADE won't be activated until the input is complete. RADE's debugging display appears as soon as the next Applesoft statement is reached.

You can also invoke RADE by using Applesoft's ampersand (&) command, commonly used by Applesoft enhancements. Since RADE is always installed before other BASIC extensions, RADE is able to trap unknown ampersand commands.

While in immediate mode, enter the ampersand (&) command now to activate RADE:

```
]&  ENTER
```

The screen is cleared and RADE displays its debugging screen, as shown on the next page.



```
 Avail Hist.    BPs   WVs    Prog. Size     Line    Statement   History
65331 ($FF33)    0     0        2 ($0002)     380        2         On

            The Real-time Applesoft Debugging Environment v1.00
                        Written by Russell E. Gibson
                    Copyright (C) 1992 Morgan Davis Group

  ➜380/2> REM Welcome to RADE!
:█
```

RADE's initial display screen.

The first time RADE presents itself, it displays its title, version number, author, and copyright information.

At the top of the screen is RADE's status bar. The status indicates the amount of memory available in the debugging history, the number of breakpoints and watch variables defined, the size of the program in memory, the current line and statement number, and the history recording mode. These features will be discussed in detail later.

Entering Commands

RADE is controlled by entering commands on the *command line,* which is preceded by a colon (:) prompt. Commands are letters or words, and are sometimes followed by arguments. Commands and arguments are separated by spaces.

A command is performed when you press ENTER. Some commands can be entered with the ⌘ key (ENTER is not required).

Enter the following command for practice:

D ENTER

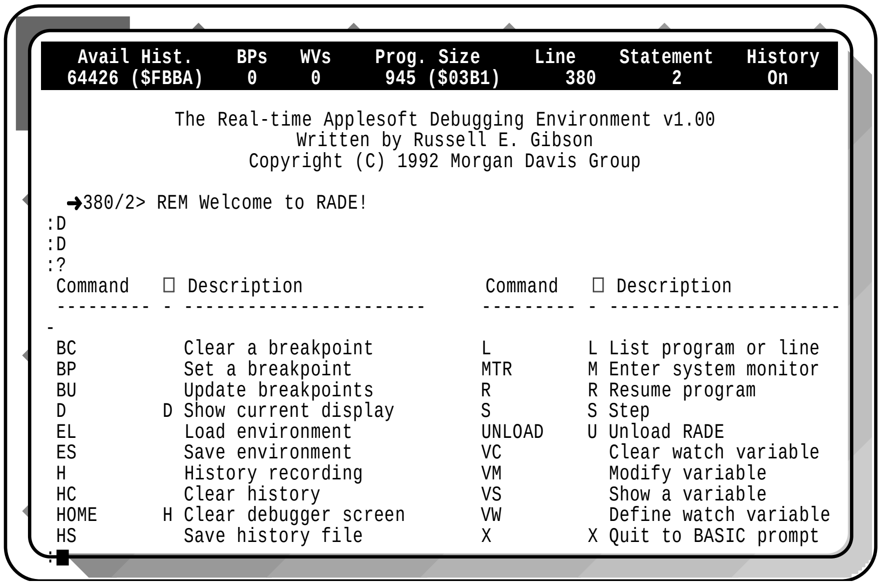
The D (display) command shows the program's screen. Typing any key switches back to RADE's debugging screen.

For practice with ⌘ keys, hold down ⌘ and press D at the same time. ⌘-D is a shortcut for entering D followed by ENTER. Any key, including ⌘-D, restores RADE's screen.

Online Help

Enter the question mark (?) to display RADE's command summary:

? ENTER



```
  Avail Hist.     BPs    WVs      Prog. Size      Line     Statement   History
 64426 ($FBBA)     0      0       945 ($03B1)      380         2          On

              The Real-time Applesoft Debugging Environment v1.00
                          Written by Russell E. Gibson
                       Copyright (C) 1992 Morgan Davis Group

  ➜380/2> REM Welcome to RADE!
:D
:D
:?
 Command   □ Description                       Command   □ Description
 --------- - ----------------------            --------- - ----------------------
-
 BC          Clear a breakpoint                L         L List program or line
 BP          Set a breakpoint                  MTR       M Enter system monitor
 BU          Update breakpoints                R         R Resume program
 D         D Show current display              S         S Step
 EL          Load environment                  UNLOAD    U Unload RADE
 ES          Save environment                  VC          Clear watch variable
 H           History recording                 VM          Modify variable
 HC          Clear history                     VS          Show a variable
 HOME      H Clear debugger screen             VW          Define watch variable
 HS          Save history file                 X         X Quit to BASIC prompt
:█
```

Built-in help summary.

Commands you can enter in the command line are listed under the Command heading.

Letters in the □ column indicate commands entered in combination with the ⌘ key.

To get a brief description of a command, type **?** followed by the command (remember to include a space between the command and its argument):

```
? VW ENTER
```

All of RADE's commands are discussed in detail in the next chapter.

Debugging Concepts

To use RADE effectively you should be familiar with these standard debugging concepts:

Stepping. This is perhaps the most useful feature of a debugger because it allows you to pause the execution of a program at each instruction. The debugger freezes the program before each statement is performed so that you can examine variables and program flow.

Tracing. Tracing a program's execution gives you a running history of program flow. Unlike stepping, each statement executed is recorded with no interruption of the program. Later, you can examine the history to study program flow.

Breakpoints. If you know the general area in a program where debugging (e.g. stepping) should begin, you can set a breakpoint there. When execution reaches the breakpoint's location, the debugger kicks in automatically.

Interruption. RADE can be activated manually during the course of a running program by pressing ⌘-CTRL-DEL or placing a lone ampersand (&) in your program. This interrupts the program as if a breakpoint where set at that point in the program.

RADE Commands

This chapter describes RADE's debugging commands, their options and syntaxes. Commands are presented in groups that relate to a particular function.

Symbols & Syntax

Arguments are separated from commands and other arguments by one space character.

Optional arguments are shown between square brackets, [like this]. Don't include the brackets.

Choices are separated by the vertical bar character (**|**). For example, AA **|** BB means you can enter AA or BB as an argument.

line denotes a line number (e.g. 2800).

stmt denotes a statement number. Since more than one statement can be given per line, RADE allows you to refer to a particular statement by its position on a line.

line/stmt is used frequently when both a *line* and *stmt* number are required. Always remember to include the slash with this format (e.g. 2800/2).

pathname is a legal ProDOS pathname, a sequence of volume, subdirectory, and file names that reference a file on a ProDOS disk.

variable is any legal variable name in Applesoft.

Breakpoints

If you know the general area in a program where debugging (e.g. stepping) should begin, you can set a breakpoint. When execution reaches the breakpoint's location, the debugger kicks in automatically.

► **BC [*line/stmt*]**

Clears a breakpoint. Without arguments, BC clears all breakpoints.

A breakpoint can be specified by a *line/stmt* reference:

```
BC 2800/2
```

It is possible to clear all breakpoints on a given *line*, by using = or * for *stmt*.

► **BP [*line/stmt*]**

(⌘-B) Sets or lists breakpoints. BP by itself displays all breakpoints. To set a breakpoint, include a *line/stmt* reference:

```
BP 2800/2
```

This triggers RADE when the 2nd statement in line 2800 is reached.

► **BU**

Breakpoint update. When RADE detects that the size of the current program has changed, it automatically updates its pointers so that they reference the right line and statement locations.

BU is diligent about doing this without intervention, however it is included for the rare occasion when modications do not affect the program’s length. This might occur when loading a new program with the same length as the previous program. Invalid breakpoints aren’t dangerous — they just keep RADE from stopping on breakpoints correctly.

History

RADE maintains a large area of memory where it stores all debugging commands and their results (except for work you do in the system monitor via RADE’s MTR command). This area is known as the debugging history buffer, which can be reviewed at any time. RADE records information until the buffer fills (about 64K of text)—then it stops. It is up to you to clear the filled buffer (perhaps saving it first) so that RADE can record new information.

► ↑

Reviews the debugging history. Use the ↑ and ↓ keys to scroll through the history, one line at a time. To browse through the history one screen at a time, hold ⌘ while pressing the ↑ or ↓ key. ESC returns to RADE’s command line.

The history can be viewed while entering a command without disturbing the command line’s contents. For example, if you are setting a breakpoint, but forgot the statement number, you might scroll through the history to find it. When done scrolling through the history, RADE allows you to complete the command line.

► H + | - | ON | OFF

History recording. Recording is enabled when RADE is installed, but can be turned off by giving the - or OFF arguments, and subsequently resumed with the + or ON arguments.

NOTE: If the history buffer fills, RADE turns off history recording automatically. Use the H command to re-enable history recording after clearing the buffer with HC.

► HC

Clears the history buffer. Use HC when the history has filled and is no longer able to record information, or when you no longer need history buffer's contents.

In addition to clearing the history, HC also clears RADE's debugging display.

► HS *pathname*

Saves history. The current history buffer is saved to file specified by *pathname*. If *pathname* refers to a file that already exists, RADE asks if you want to append to the file. If you don't want to append, RADE asks if you want to replace the file.

The file is stored in standard ASCII text format, suitable for printing. RADE does not clear the history after saving it.

Environment Files

RADE's environment files are excellent for recording debugging sessions for programs in development. With the ability to set breakpoints and watch variables for use with a particular program, it is convenient to store these settings for easy recall. Environment files record breakpoints, watch variables, history buffer, and the history recording state.

► **`EL`** ***`pathname`***

Loads an environment file created by the ES command.

► **`ES`** ***`pathname`***

Saves the environment to a file. The environment can be restored later using the EL command. It is suggested that environment files possess the same name as the program file with a .env suffix (e.g. ANIMALS.ENV).

Program Flow

RADE's program flow features allow you to take a snapshot of your program after each statement and view the results. They also allow you to divert program flow as if a GOTO were placed in your program. You can even halt a program, exit to Applesoft's immediate mode, then continue again where you left off.

► **`R [`** ***`line`*** **`]`**

(⌘-R) Resume execution of the program. Without arguments, the original program's display is restored and program execution continues.

The *line* argument allows you to resume execution at a particular line in the program, effectively performing a GOTO to the line specified.

```
     R 6502
```

This would resume program execution at line 6502.

► **S [*n*]**

(⌘-S) Steps through statements. After the statement is executed, RADE activates itself prior to the execution of the next statement. When stepping, the current statement is displayed along with any variables being watched.

You can trace through several statments at once by specifying *n* as the number of statements to step.

► **X**

(⌘-X) Exits RADE, stopping the BASIC program, so that you access Applesoft's immediate mode.

NOTE: Applesoft's CONT command can resume the program from the point where RADE's X command interrupted it.

Variables

RADE includes a number of commands that make inspecting variables a snap. You can instruct RADE to display a particular set of variables and their values each time RADE's debugging screen is invoked (called *watch variables*). You can view the contents of any variable, list the names of variables that have been defined, and even change their values.

► **VC [*variable*]**

Clear watch variable. This removes a variable from RADE's variable watching list. If VC is entered by itself, all watch variables are cleared.

► **VM *variable value***

Modify variable. This assigns a *value* to the specified *variable* which must exist in memory. The *value* must be a string or numeric constant—no variables or expressions are accepted. Examples:

```
VM A$ "Test"
VM X(4,2) 5
```

For floating point variables, exponents are not allowed, though simple decimal notation can be used.

► **VS [*variable*]**

Show variable. Displays the value of the specified *variable*. Examples:

```
:VS A$
      A$ = "Test"
:VS X(4,2)
      X() = 5
```

VS reports undefined variables ("does not exist"), displaying zero for numeric variables and "" for string variables.

Without arguments, VS lists the names of all variables that have been assigned values.

► **VW [*variable*]**

(⌘ -V) Watch variable. Each time RADE is activated, all variables being watched are automatically displayed. Up to eight watch variables can be defined at one time.

Without arguments, VW displays all watch variables and their contents.

NOTE: VW does not support array variables.

Other Commands

Miscellaneous RADE commands include the following:

► **? [*command*]**

(⌘-/) Displays help. If a *command* argument is specified, a brief description of that command is presented. Cmd-/ (or ? with no arguments) displays a summary of RADE's commands.

► **D**

(⌘-D) Displays the current program screen as it was just before entering RADE. Press any key to return to RADE's debugging display.

► **HOME**

(⌘-H) Clears the debugging screen. This command does not affect the debugging history.

► L [.] | [[[*line*] -] *line*]

(⌘-L) List program lines. Without arguments, the entire program is listed.

The period (**.**) signifies the current program line—that is, the line currently interrupted by RADE. When RADE is invoked from immediate mode, the period lists the last line executed.

A range of lines may be specified. Use a starting *line*, a dash, and optionally an ending *line*.

If the listing includes the current statement, an arrow (➔) is displayed. If a statement listed has a breakpoint, a diamond (◆) is shown with the breakpoint's index number.

Lines are displayed with each statement identified by its statement number. Example:

```
:L 200
◆1   200/1>  PRINT "Testing"
     200/2> X = 25
     200/3>  FOR I = 1 TO X
    ➔200/4>  PRINT I
     200/5>  NEXT
```

► MTR

(⌘-M) System monitor. Use CTRL-Y ENTER to return to RADE when done.

▶ UNLOAD

(⌘-U) Removes RADE from memory. RADE disconnects itself from the Apple IIGS keyboard flush vector, restores the /RAM volume, and releases the 768 bytes of main memory it occupied.

If ⌘-U is used, RADE asks if you really want to unload it. If so, type Y.

NOTE: RADE automatically unloads itself when the ProDOS BASIC ***BYE*** *command is issued.*

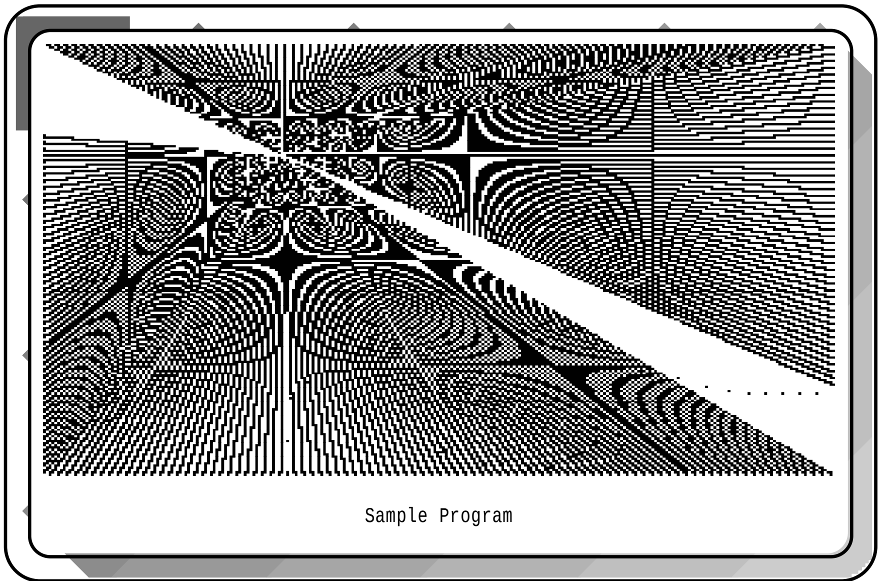
A Debugging Session

This chapter walks you through a typical debugging session using most of the commands described in the previous chapter. The diskette comes with a program named FIX.ME, riddled with bugs. Follow along to repair FIX.ME while gaining hands-on experience with RADE.

About FIX.ME

The FIX.ME program (when bug-free) turns on the Apple's high-resolution graphics display and draws an interesting pattern, as shown below.

FIX.ME (after you fix it).



The program continues to cycle through random displays, pausing briefly after each one, until you press a key.

Running FIX.ME

If RADE isn’t already in memory, restart the RADE diskette and enter into Applesoft. Otherwise, make sure the current prefix is set to the RADE disk.

Next, run the FIX.ME program:

```
]RUN FIX.ME   ENTER
```

After loading, the screen immediately clears, then

```
Sample Program Finished
```

is printed. And you return to immediate mode. This program is supposed to display a pretty graphic design, but something is definitely not right. Even worse, the program quit without doing a thing, and didn’t even have the courtesy of displaying an error message.

Use RADE to step through the program to help locate and repair the mistakes that prevent FIX.ME from running correctly.

Stepping

First, invoke RADE by typing ⌘-CTRL-DEL. Now, RUN the program again by typing RUN. RADE activates and shows the first line in the program:

```
→100/1> ONERR GOTO 190
```

Step through the first statement using the ⌘-S keys. This statement sets up error handling with line 190. The next statement is shown:

```
→110/1> HOME
```

Step again and the screen clears.
The next statement to execute is:

```
➜120/1> VTAB 25
```

Uh oh. Before you step again, notice the value being used with VTAB. The Apple II display has only 24 lines. Surely, that will cause an error. And since error handling is turned on, the program is going to transfer execution to the handler in line 190.

Listing Lines

Use the L command to list a few lines starting with 190 so you can see what is happening when an error occurs:

```
      :L 190-230
         190/1> POKE 49168,0
         200/1> HOME
         210/1> TEXT
         220/1> PRINT "Sample Program
Finished"
         230/1> END
```

That explains why the program just clears the screen, prints a message, and quits without doing much else. That VTAB statement needs to be fixed.

Exiting

To quickly exit RADE and stop the program from running, using the ⌘-X keys. Your screen shows:

```
BREAK IN 120
]■
```

Now you can repair the bogus VTAB. In line 120, change the VTAB 25 to VTAB 24.

Disabling ONERR

While debugging with RADE, it is sometimes helpful to turn off Applesoft's ONERR handling, as illustrated by the first bug in FIX.ME. Had Applesoft's error handling been turned off, the VTAB 25 would have generated an error message and stopped the program instantly.

With this insight, insert a REM statement at the beginning of line 100 to effectively disable ONERR handling. You can always remove the REM to enable it later. (A program can selectively shut off Applesoft's error handling by using POKE to store zero at location 216.)

Using Breakpoints

RUN the program once more. This time the graphics screen is obviously invoked, and the program appears to be drawing some kind of design to the screen. But another error is causing it to quit early:

```
?ILLEGAL QUANTITY ERROR IN 320
]■
```

(Your display may show that the error occurs in line 300. If that's the case, substitute 300 for each reference to line 320 throughout the remainder of this chapter.)

Now is a good time to set a breakpoint. Type ⌘-CTRL-DEL and then RUN the program again. When RADE comes up, enter:

```
:BP 320/1   ENTER
```

This places a breakpoint on the first statement in line 320. Notice how the status bar at the top of the screen now shows 1 under the BPs heading. Each time Applesoft attempts to execute line 320, RADE will be invoked automatically.

Resuming Execution

Resume program execution using ⌘-R. Almost immediately, we're back in RADE at line 320.

```
◆1➔320/1> HPLOT X + 1,0 TO
          XCENTER,YCENTER TO
          279 - X - 1, 191
```

Line 320 hasn't executed yet, but we know that the HPLOT command is getting upset about one of the values being passed to it. (Hint: HPLOT allows coordinate points to be used from 0,0 to 279,191.)

Showing Variables

Inspect the values of the variables used in line 320 to see if they're in range:

```
:VS X   ENTER
   X = 0
:VS XCENTER  ENTER
   XC = 135.819372
:VS YCENTER  ENTER
   YC = 52.6188322
```

The values shown on your screen for XCENTER and YCENTER probably differ since the program generates these values randomly. In any case, the values are within the range that HPLOT allows, so stepping once ought to work without any trouble.

Type ⌘-S to step once. No errors? So far, so good.

Switching Displays

Type ⌘-D to view the program's display. The screen shows two rays extending from a random center point out to the edges of the graphics screen. Press any key to return to RADE's debugging display.

Apparently, the error doesn't occur everytime, but only when one of the variables is out of range. From within RADE, look at the program listing to put the error into context by typing ⌘-L. This lists the entire program.

The listing indicates the current program line as being 320. You can see that 320 is inside of a FOR-NEXT loop that increments X.

Watching Variables

Use the VW command to watch the variable X:

```
:VW X   ENTER
```

Each time RADE comes up, it displays the current line and the value in X.

Now use ⌘-R to resume execution through another loop. RADE returns instantly because of the breakpoint in line 320. Press ⌘-R a few more times.

It may take some time, and hundreds of ⌘-R keys, until the error manifests itself again. Forunately, there is an easier way to trace through the program non-stop without any intervention on your part.

Don't wear out your fingers now—read on.

Tracing

You know what the trouble is, but it takes time before the bug bites again. The best thing to do in this case is to allow RADE to rapidly step through each statement on its own until the program bombs again. Enter the following:

```
:H OFF   ENTER
```

This turns off history recording because the next step(s) will generate a lot of debugger output.

Next, enter:

```
:S 999   ENTER
```

This tells RADE to perform 999 steps, a suffcient number to speed your way to the error. At some point the program will crash and you'll be able to look at the last line executed before the error occurred.

Invoking RADE with &

Eventually, the program does end up in immediate mode with an ILLEGAL QUANTITY ERROR. Return to RADE by entering an ampersand (&) by itself:

```
]&  ENTER
```

The value of X is shown (since we're watching it) along with the last statement executed. Ah ha! HPLOT's horizontal coordinate in X is out of range!

Before continuing, turn history recording back on:

```
:H ON  ENTER
```

RADE will now continue to record your debugging operations.

Viewing the History

Use ↑ to scroll backward through RADE's history and stop when you see the program listing generated with ⌘-L earlier.

Locate the FOR loop that begins in line 280. Notice how it affects X by looping from 0 to 280! While the graphics display has 280 horizontal pixels, the ending value for the loop should be 280 minus SIZE, the amount that X is incremented in each pass.

Press ESC to exit the history viewing mode. Then use the R command to return to immediate mode. Change line 280 to:

```
280  FOR X = 0 TO 280 - SIZE STEP SIZE
```

This ensures that X stays within the horizontal pixel boundary.

Clearing Breakpoints

RUN the program again. Oops, we're back in RADE again at line 320 —that breakpoint is still set. Remove it with the BC command:

```
:BC 320/1   ENTER
```

Now type ⌘-R to resume execution. The program cycles through a few screens, drawing a design, pausing for a moment, and starting a new screen.

Something still isn't right. After the pause, you can hear a faint click from the speaker. And, if left running long enough, the program stops on its own. (It is supposed to keep cycling until you press a key to stop it—and that doesn't appear to work either).

Inspecting Variables

This time, RUN the program first to allow the program to define its variables.

Now type ⌘-CTRL-DEL to break into RADE. To see which variables are currently defined use VS:

```
:VS    ENTER
Simple variables:
      KE  XC  YC  SI  X

Array variables:
      None
```

You're familiar with all the above except for KE (KEY in the program listing). Take a look at KE directly so you can discover its contents:

```
:VS KE    ENTER
   KE = 49200
```

Location 49200 in peripheral memory corresponds to the speaker location ($C030). Any access to that location will click the speaker.   List line 180:

```
180/1> IF PEEK (KEY) < 128 THEN 150
```

KEY is accessing the speaker when, presumably, it should be referencing the keyboard input location 49152. This explains the clicks, and may also be the reason why a keypress doesn't halt the program.

NOTE: Applesoft stores only the first two letters of a variable name in memory. Your program listings may use longer names, but RADE will only see their first two letters.

Modifying Variables

For testing purposes, change the value in KEY to 49152 (the keyboard input location) instead of 49200 (the speaker location).

```
:VM KEY 49152   ENTER
```

VM modifies the variable KEY (recognized as KE by Applesoft) to contain 49152.

Resume execution again with ⌘-R. This time, the clicking has gone away. And the program will run forever until a key is pressed.

Final Fix

The program is still not completely right, because the modification to KEY was made in memory only. The program still assigns 49200 to KEY in line 135. You’ll want to change it to:

```
135  KEY = 49152
```

Congratulations! With RADE, you’ve successfully debugged FIX.ME.

(You can save the new program as MOIRE.)

RADE Command Chart

Command	**⌘ Key**	**Description**
↑		History
? [*command*]	⌘-/	Help
BC [*line / stmt*]		Clear breakpoint
BP [*line / stmt*]	⌘-B	Set breakpoint
BU		Update breakpoints
D	⌘-D	Switch display modes
EL *pathname*		Load environment
ES *pathname*		Save environment
H + **|** - **|** ON **|** OFF		History recording
HC		Clear history
HOME	⌘-H	Clear debugging display
HS *pathname*		Save history
L [.] **|** [[[*line*] -] *line*]	⌘-L	List program lines
MTR	⌘-M	Enter system monitor
R [*line*]	⌘-R	Resume program
S [*n*]	⌘-S	Step through statement(s)
UNLOAD	⌘-U	Unload and remove RADE
VC [*variable*]		Clear a watch variable
VM *variable value*		Modify a variable
VS [*variable*]		Show variable(s)
VW [*variable*]	⌘-V	Watch a variable
X	⌘-X	Exit to immediate mode

ASCII Chart

Low				High	Low				High	Low					High	Low				High
0	$00	^@	128	$80	32	$20	SPC	160	$A0	64	$40	@		192	$C0	96	$60	‘	224	$E0
1	$01	^A	129	$81	33	$21	!	161	$A1	65	$41	A		193	$C1	97	$61	a	225	$E1
2	$02	^B	130	$82	34	$22	"	162	$A2	66	$42	B	◣	194	$C2	98	$62	b	226	$E2
3	$03	^C	131	$83	35	$23	#	163	$A3	67	$43	C	⧗	195	$C3	99	$63	c	227	$E3
4	$04	^D	132	$84	36	$24	$	164	$A4	68	$44	D	✓	196	$C4	100	$64	d	228	$E4
5	$05	^E	133	$85	37	$25	%	165	$A5	69	$45	E	☑	197	$C5	101	$65	e	229	$E5
6	$06	^F	134	$86	38	$26	&	166	$A6	70	$46	F	⏎	198	$C6	102	$66	f	230	$E6
7	$07	^G	135	$87	39	$27	'	167	$A7	71	$47	G	≣	199	$C7	103	$67	g	231	$E7
8	$08	^H	136	$88	40	$28	(	168	$A8	72	$48	H	←	200	$C8	104	$68	h	232	$E8
9	$09	^I	137	$89	41	$29	)	169	$A9	73	$49	I	…	201	$C9	105	$69	i	233	$E9
10	$0A	^J	138	$8A	42	$2A	*	170	$AA	74	$4A	J	↓	202	$CA	106	$6A	j	234	$EA
11	$0B	^K	139	$8B	43	$2B	+	171	$AB	75	$4B	K	↑	203	$CB	107	$6B	k	235	$EB
12	$0C	^L	140	$8C	44	$2C	,	172	$AC	76	$4C	L	‾	204	$CC	108	$6C	l	236	$EC
13	$0D	^M	141	$8D	45	$2D	-	173	$AD	77	$4D	M	↵	205	$CD	109	$6D	m	237	$ED
14	$0E	^N	142	$8E	46	$2E	.	174	$AE	78	$4E	N	█	206	$CE	110	$6E	n	238	$EE
15	$0F	^O	143	$8F	47	$2F	/	175	$AF	79	$4F	O	⇤	207	$CF	111	$6F	o	239	$EF
16	$10	^P	144	$90	48	$30	0	176	$B0	80	$50	P	⇥	208	$D0	112	$70	p	240	$F0
17	$11	^Q	145	$91	49	$31	1	177	$B1	81	$51	Q	⤓	209	$D1	113	$71	q	241	$F1
18	$12	^R	146	$92	50	$32	2	178	$B2	82	$52	R	⤒	210	$D2	114	$72	r	242	$F2
19	$13	^S	147	$93	51	$33	3	179	$B3	83	$53	S	─	211	$D3	115	$73	s	243	$F3
20	$14	^T	148	$94	52	$34	4	180	$B4	84	$54	T	└	212	$D4	116	$74	t	244	$F4
21	$15	^U	149	$95	53	$35	5	181	$B5	85	$55	U	→	213	$D5	117	$75	u	245	$F5
22	$16	^V	150	$96	54	$36	6	182	$B6	86	$56	V	▒	214	$D6	118	$76	v	246	$F6
23	$17	^W	151	$97	55	$37	7	183	$B7	87	$57	W	▒	215	$D7	119	$77	w	247	$F7
24	$18	^X	152	$98	56	$38	8	184	$B8	88	$58	X	⊏	216	$D8	120	$78	x	248	$F8
25	$19	^Y	153	$99	57	$39	9	185	$B9	89	$59	Y	⊐	217	$D9	121	$79	y	249	$F9
26	$1A	^Z	154	$9A	58	$3A	:	186	$BA	90	$5A	Z	▕	218	$DA	122	$7A	z	250	$FA
27	$1B	^[	155	$9B	59	$3B	;	187	$BB	91	$5B	[	◆	219	$DB	123	$7B	{	251	$FB
28	$1C	^\	156	$9C	60	$3C	<	188	$BC	92	$5C	\	⚌	220	$DC	124	$7C	|	252	$FC
29	$1D	^]	157	$9D	61	$3D	=	189	$BD	93	$5D	]	╪	221	$DD	125	$7D	}	253	$FD
30	$1E	^^	158	$9E	62	$3E	>	190	$BE	94	$5E	^	⊡	222	$DE	126	$7E	~	254	$FE
31	$1F	^_	159	$9F	63	$3F	?	191	$BF	95	$5F	_	▏	223	$DF	127	$7F	DEL	255	$FF
Low				High	Low				High	Low					High	Low				High

ProDOS File Types

Type	Hex	Dec	Description
UNK	$00	0	Unknown
BAD	$01	1	Bad Blocks
PCD	$02	2	Apple /// Pascal Code
PTX	$03	3	Apple /// Pascal Text
TXT	$04	4	ASCII Text
PDA	$05	5	Apple /// Pascal Data
BIN	$06	6	General Binary
FNT	$07	7	Apple /// Font
FOT	$08	8	Graphics
BA3	$09	9	Apple /// BASIC Program
DA3	$0A	10	Apple /// BASIC Data
WPF	$0B	11	Word Processor
SOS	$0C	12	Apple /// SOS System
DIR	$0F	15	Folder
RPD	$10	16	Apple /// RPS Data
RPI	$11	17	Apple /// RPS Index
AFD	$12	18	Apple /// AppleFile Discard
AFM	$13	19	Apple /// AppleFile Model
AFR	$14	20	Apple /// AppleFile Report Format
SCL	$15	21	Apple /// Screen Library
PFS	$16	22	PFS Document
ADB	$19	25	AppleWorks Data Base
AWP	$1A	26	AppleWorks Word Processor
ASP	$1B	27	AppleWorks Spread Sheet
TDM	$20	32	Desktop Manager Document
8SC	$29	42	Apple II Source Code
8OB	$2A	43	Apple II Object Code
8IC	$2B	44	Apple II Interpreted Code
8LD	$2C	45	Apple II Language Data
P8C	$2D	46	ProDOS 8 Code Module
FTD	$42	66	File Type Names
GWP	$50	80	Apple IIGS Word Processor
GSS	$51	81	Apple IIGS Spread Sheet
GDB	$52	82	Apple IIGS Data Base
DRW	$53	83	Drawing
GDP	$54	84	Desktop Publishing
HMD	$55	85	Hypermedia
EDU	$56	86	Educational Data
STN	$57	87	Stationery
HLP	$58	88	Help
COM	$59	89	Communications
CFG	$5A	90	Configuration
ANM	$5B	91	Animation
MUM	$5C	92	Multimedia
ENT	$5D	93	Entertainment
DVU	$5E	94	Development Utility

Continued . . .

ProDOS File Types (Continued)

Type	Hex	Dec	Description
BIO	$6B	107	PC Transporter BIOS
TDR	$6D	109	PC Transporter Driver
PRE	$6E	110	PC Transporter Pre-Boot
HDV	$6F	111	PC Transporter Volume
WP	$A0	160	WordPerfect Document
GSB	$AB	171	Apple IIGS BASIC Program
TDF	$AC	172	Apple IIGS BASIC TDF
BDF	$AD	173	Apple IIGS BASIC Data
SRC	$B0	176	Apple IIGS Source
OBJ	$B1	177	Apple IIGS Object
LIB	$B2	178	Apple IIGS Library
S16	$B3	179	GS/OS Application
RTL	$B4	180	GS/OS Run-time Library
EXE	$B5	181	GS/OS Shell Application
PIF	$B6	182	Permanent Initialization
TIF	$B7	183	Temporary Initialization
NDA	$B8	184	New Desk Accessory
CDA	$B9	185	Classic Desk Accessory
TOL	$BA	186	Tool
DRV	$BB	187	Device Driver
LDF	$BC	188	Load File
FST	$BD	189	GS/OS File System Translater
DOC	$BF	191	GS/OS Document
PNT	$C0	192	Packed Super Hi-Res Picture
PIC	$C1	193	Super Hi-Res Picture
ANI	$C2	194	Animation
PAL	$C3	195	Palette
OOG	$C5	197	Object Oriented Graphics
SCR	$C6	198	Script
CDV	$C7	199	Control Panel
FON	$C8	200	Font
FND	$C9	201	Finder Data
ICN	$CA	202	Icons
MUS	$D5	213	Music Sequence
INS	$D6	214	Instrument
MDI	$D7	215	MIDI
SND	$D8	216	Sampled Sound
DBM	$DB	219	Relational Data Base File
LBR	$E0	224	Archival Library
ATK	$E2	226	AppleTalk Data
R16	$EE	238	EDASM 816 Relocatable File
PAS	$EF	239	Pascal Area
CMD	$F0	240	BASIC Command
LNK	$F8	248	EDASM Linker
OS	$F9	249	GS/OS System File
INT	$FA	250	Integer BASIC Program
IVR	$FB	251	Integer BASIC Variables
BAS	$FC	252	Applesoft BASIC Program
VAR	$FD	253	Applesoft BASIC Variables
REL	$FE	254	Relocatable Code
SYS	$FF	255	ProDOS 8 System Application

Error Codes

0 **NEXT Without FOR**: a NEXT was encountered without a matching FOR.
2 **Range Error**: an invalid argument value was specified.
3 **No Device Connected**: the given slot has no disk drive installed.
4 **Write Protected Disk**: unable save data unless write-enabled.
5 **End of Data**: an attempt was made to read data past the end of a file.
6 **Path Not Found**: the path to a filename was not found.
7 **File Not Found**: the specified file was not found.
8 **I/O Error**: the drive went offline or the disk has a media defect.
9 **Disk Full**: no room exists on the disk storing more data.
10 **File Locked**: the file is protected against modification or removal.
11 **Invalid Option**: an option not allowed for a certain command was used.
12 **No Buffers Available**: not enough memory for further disk functions.
13 **File Type Mismatch**: an invalid attempt was made to access a special file.
14 **Program Too Large**: you've written a FAT and SLOPPY program.
15 **Not Direct Command**: command was issued from immediate mode.
16 **Syntax Error**: a filename is illegal or a program statement misspelled.
17 **Directory Full**: the root volume contains too many filenames.
18 **File Not Open**: an attempt was made to read or write from an closed file.
19 **Duplicate File Name**: a RENAME or CREATE used on an existing file.
20 **File Busy**: an attempt to re-OPEN or modify an OPEN file.
21 **File Still Open**: upon entering immediate mode, a file was found OPEN.
22 **RETURN Without GOSUB**: a RETURN with no matching GOSUB.
42 **Out of Data**: an attempt was made to READ past the last DATA item.
53 **Illegal Quantity**: an out-of-range value was used with a certain command.
69 **Overflow**: you used an awfully BIG or amazingly SMALL number.
77 **Out of Memory**: program code and variables have used up free memory.
90 **Undef'd Statement**: a line number which does not exist was referenced.
107 **Bad Subscript**: an array subscript is larger than the given DIMension.
120 **Redim'd Array**: an attempt was made to reDIMension an existing array.
133 **Division by Zero**: division by zero is undefined (remember your algebra?)
163 **Type Mismatch**: a numeric or string value was used incorrectly.
176 **String Too Long**: the given string was larger than was allowed.
191 **Formula Too Complex**: go easy on the machine, Einstein.
224 **Undef'd Function**: reference to an undefined FuNction was made.
254 **Reenter**: user input was not of the type or format required.
255 **Control-C Interrupt**: CTRL -C was pressed.

Index

NOTES

NOTES

BASIC PEST CONTROL KILLS BUGS DEAD!

When bugs invade your BASIC programs, reach for RADE, the Real-time Applesoft Debugging Environment! Pressing ⌘ - CTRL - DEL halts any program to enter RADE's powerful debugging mode. Explore a program's inner workings without disturbring memory or the screen display. With RADE's stealth-like features, bugs become an endangered species!

- ▶ Breakpoints
- ▶ Variable monitoring
- ▶ Stepping
- ▶ Tracing
- ▶ Debugging history buffer
- ▶ Environment files
- ▶ Run-time variable modification
- ▶ Transparent to programs
- ▶ Uses 768 bytes of main memory
- ▶ Program flow modification
- ▶ Variable name listing
- ▶ Preserves the program's display
- ▶ List program lines while running
- ▶ Run-time access to the monitor
- ▶ Built-in help
- ▶ Easy and fun to use

Morgan Davis Group
10079 Nuerto Lane • Rancho San Diego • CA 91977-7132 • (619) 670-0563