VoodooMonkey Notes

Mike Lockwood
April 15, 1993
(revised August 10, 1993)

Disclaimer

VoodooMonkey is a source level debugger compatible with MPW. VoodooMonkey is an experimental prototype developed at Apple Computer, Inc. It is not a supported product of Apple. VoodooMonkey might not be compatible with future Apple hardware and system software.

VoodooMonkey has not undergone any formal testing or quality assurance. You will undoubtedly encounter bugs while using VoodooMonkey, some that might crash your computer. Use at your own risk!!!

Requirements

In order to run, VoodooMonkey requires that you install the DebuggerINIT in your System Folder. VoodooMonkey requires System 7.0 or better, a 68020 or better, and Color Quickdraw. VoodooMonkey provides support for debugging applications that use Thread Manager 1.1. However, you don't need the Thread Manager to use VoodooMonkey.

VoodooMonkey can be used to debug programs built with MPW 3.2 or later. You must build your program with -sym on, so the Linker produces a Sym file. VoodooMonkey cannot understand the ILink StateFile, so if you use ILink, you must run the ILinkToSym tool to create a Sym file.

VoodooMonkey Features

VoodooMonkey differs from SourceBug in two major ways:

- VoodooMonkey is faster (only a few seconds to open the browser and evaluating variables is faster)

- VoodooMonkey can debug non-applications (DAs, cdevs, MPW Tools, code resources, extensions, ASLM shared libraries, etc.)

In order to achieve better performance, some browsing features were sacrificed. In particular, the browser does not browse by class and methods, like SourceBug, but only browses by file and function.

Other cool features include:

- Dynamic typing of both Pascal and C++ objects (that is, if you evaluate a variable declared to be a TEventHandler, VoodooMonkey will look at the ClassID (for PascalObject) or the vtable pointer (for C++ objects) to determine the actual type of the object and cast it to the actual subclass of the declared class, like TCheckBox.

- Allows debugging an application with multiple SYM files. For example, you could debug both your application and a plug-in code resource simultaneously, and the stack crawl window will show the call chain through your application and your plug-in.

- Support for debugging applications with multiple threads. Using the Process Browser, you can open multiple stack crawl windows, one for each thread.

- Finder-like hierarchical view of variables

- Inline editing support for modification of variable values

- Casting of variables to different types

- VoodooMonkey can start debugging a program that is already running. You can also quit VoodooMonkey without quitting the program you were debugging. To start debugging an already running application, you can either open the Sym file for the application and execute the "Run" menu command, or you can select the application in the Process Browser and use the "Target <application>" menu command.

- Change the program counter by dragging the arrow in the source code view. This allows you to skip over a function call or a range of code, or back up and execute the code again.

- "Go Until" feature. Double clicking on a breakpoint diamond resumes execution until that point in the code.

- Execution timing in microseconds. If you turn on execution timing, every time you "Run" or "Go Until", VoodooMonkey logs the number of microseconds it took to execute the code to the Log Window. The cool friend of mine that gave me the code to do this claims the timings are accurate to approximately 50 or 100 microseconds.

- A "Stop <application>" command, which stops a running application after the next event call.

- Object inspector support for MacApp 3.1a1.

Debugging Multithreaded Applications

VoodooMonkey provides debugging support for Apple's new Threads Manager. The main feature supporting threads is the ability to browse all the threads for your application in the Process Browser and open separate stack crawl windows for each thread. This allows you to examine the state of all your threads, and can be particularly helpful when debugging synchronization and deadlock problems.

To open a stack crawl for a thread, your application must first be stopped (for example, at a breakpoint, or you could use the "Stop <application>" menu command. Open the Process Browser, and choose your application in the Process list. All the threads for your application will appear in the Threads list. Select a thread, and choose the "Open Stack Crawl for Thread" menu item in the Process menu.

Preemptive threads can be scheduled at interrupt time, so it is not safe to stop the application and switch to VoodooMonkey when the current thread is preemptive. To protect against this danger, VoodooMonkey ignores breakpoints that are hit in a preemptive thread. If your application crashes in a preemptive thread, your low-level debugger (MacsBug, TMON or Jasik) will handle the crash instead of VoodooMonkey.

Using VoodooMonkey with MacApp 3.1a1

VoodooMonkey contains support for an object inspector similar to SourceBug's that is compatible with MacApp 3.1a1. In order to use this feature, you need to build your MacApp application with the -Inspector option. Note that this feature needs special changes made to MacApp 3.1a1, so the object inspector is not compatible with earlier versions of MacApp.

To open an object inspector, choose the "New Inspector Window" item in the "Inspect" menu. VoodooMonkey only tracks objects created while your MacApp application is targeted. If you target an already running MacApp application, the inspector will only show objects created since you targeted the program.

VoodooMonkey uses MultiFinder temporary memory to store the object lists to support the inspector. You may have problems if you don't have any free memory in MultiFinder's heap. However, if you were able to successfully link your MacApp application, you probably have the extra memory!

Using VoodooMonkey with Other Debuggers

VoodooMonkey has various levels of compatibility with other debuggers. You can run VoodooMonkey with MacsBug, TMON, or Jasik's Debugger installed. While you are debugging with VoodooMonkey, you can switch to debugging with MacsBug, TMON, or Jasik by using the "Switch to Low-level Debugger" menu command. If you are using

TMON, you can switch back to VoodooMonkey using the TMON macro that comes with SourceBug. See the file "Macro for TMON Professional" that ships with SourceBug for more details.

VoodooMonkey and SourceBug 1.1 can run simultaneously. That is, you can use SourceBug 1.1 to debug one program while you are using VoodooMonkey to debug another. You can use Sade safely with the DebuggerINIT installed, but can not use VoodooMonkey while Sade is running.

Debugging Non-Applications

Debugging non-applications, such as code resources, MPW Tools, and shared libraries, is fairly straight forward. All you need to do is open a browser for the Sym file for your program, and set a breakpoint. Then, do what ever you need to do to run this code. When you hit the breakpoint, VoodooMonkey will create a stack crawl window for the application that is executing the non-application's code and stops that application's process. For example, VoodooMonkey will suspend the Finder if you are debugging a control panel, or HyperCard if you are debugging an XCMD.

If you are the developer of the application that is executing the code, you can open the Sym file for the application and debug both the application and extension simultaneously. If VoodooMonkey finds a Sym file for the application in the same directory as the application, VoodooMonkey will do this for you automatically.

See the "Gotchas" below for more details for common problems with debugging non-applications, and how to fix them.

Common VoodooMonkey Gotchas

1) If you are developing a code resource, shared library, control panel, or other non-application program, you might need to copy the program to another location, like the "Extensions" folder. If you do this, it is usually a good idea to copy the Sym file as well. Otherwise, if you open the Sym file in the original directory and set breakpoints, you will be setting breakpoints in the copy of your program that is in the same directory as the Sym file, rather than the copy you are actually running. If this happens, the breakpoints will not be set properly.

2) If you write a code resource that is later copied (with ResEdit or Rez) into another resource fork (like an XCMD in a HyperCard stack), You need to make sure that the Sym file that you open with VoodooMonkey is NOT in the same directory as the resource file that the linker originally created. For example, if you built "MyXCMD.rsrc" and copied the XCMD resource into "MyStack", you would need to make sure that "MyXCMD.rsrc.SYM" is NOT in the same directory as "MyXCMD.rsrc". Otherwise, VoodooMonkey will automatically target "MyXCMD.rsrc" instead of "MyStack". You should make sure the Sym file is in a different location. Then, when you open the Sym file,

VoodooMonkey won't find the file created by the linker, and will prompt you to find "MyXCMD.rsrc". Instead of selecting "MyXCMD.rsrc" in standard file, choose "MyStack". Then VoodooMonkey will set breakpoints in the copy of the XCMD that you will actually be running.

3) Some people have had a different problem with breakpoints being ignored. If you build an extension and then run a tool that changes the resource type of the code segments from 'CODE' to something else, VoodooMonkey will not be able to debug those code segments. This is because the Sym file says that the resource type is 'CODE' and VoodooMonkey has no way of knowing that you changed the resource type. If you absolutely must change the resource type of your resources, you will also have to write a tool that fixes up the information in the Sym file.

4) Sometimes the name of the Sym file matches an intermediate resource file, rather than the actual executable. For example, when building ASLM libraries, an intermediate file named MyLibrary.RSRC is created for a shared library called MyLibrary. The problem is that the name of the Sym file is MyLibrary.RSRC.SYM, so VoodooMonkey thinks you are trying to debug the intermediate file. The trick is to rename the Sym file to MyLibrary.SYM.