Spotlight Memory Debugger

Version 1.0

User Manual

Document Rev: 0100.1

Spotlight
Memory Debugger
v1.0

Product Design
and Engineering: Andrew Barry, Brooks Bell, Devon Hubbard

Onyx Technology, Inc.
7811 27th Ave West
Bradenton, FL 34209

(941) 795-7801 voice
(941) 795-5901 fax

Technical Support:
support@onyx-tech.com

Sales Information:
sales@onyx-tech.com

<http://www.onyx-tech.com/>

Licenses, Copyrights, and Trademarks

Table of Contents

1 Introduction

Spotlight is a tool to enable PowerPC Macintosh developers to automatically detect potentially serious errors in their code - before they have a chance to cause any damage. This includes entire classes of subtle errors that otherwise very difficult to reproduce - and Spotlight can detect them without even having to recompile your application.

The core functionality that Spotlight provides is based on a very refined form of memory protection. Future versions of the MacOS will be providing very simple memory protection, with programs being disallowed from read/writing into the memory spaces of other applications - but provide no protection for the application corrupting itself. Any bugs in code that corrupts the application heap will continue to go undetected under all future versions of the MacOS. See the Memory Protection section for more information about the memory protection that Spotlight provides.

Spotlight also provides strong validation of the parameters and return values of many Toolbox routines. These checks also extend to verify that the proper machine 'state' exists for these calls to proceed. Spotlight verifies that if you try drawing to a GWorld that it has been locked. See the Toolbox Validation section for further description of the types of errors detected.

The final facility that Spotlight provides is for tracking leaked memory. When you've completed running your application under Spotlight, it gives a list of all of the memory blocks your application has leaked, including distinguishing between Ptr/malloc/C++ object blocks, as well as a symbolic stack trace at the point that the block was allocated.

2 Installation

Installing Spotlight is a simple process.

1. Launch the Spotlight Installer.
2. Click ‘Continue’ at the introductory splash screen.
3. Read the Spotlight License Agreement and press ‘Accept’ to continue.
4. You will then be presented with the Easy Install dialog, shown below in figure 1.

Figure 1

5. Click ‘Install’ to perform an easy install, or select ‘Custom Install’ from the popup menu in the upper left to select a specific package to install.
6. Once installation is complete you are ready to being using Spotlight. No system restart is necessary.

System Requirements

Spotlight requires a PowerPC Macintosh to operate and instrumentation and testing is performed on PowerPC executables. Minimum 16mb of physical RAM. Minimal hard disk space is required other than what your own projects occupies.

3 The Basics

Spotlight performs the following tests on your PowerPC code.

- Memory protection with validation on all read/write accesses.
- Leaks Detection and reporting
- Toolbox parameter checking (also known as Discipline).

Memory Protection

The memory protection that Spotlight provides is far more subtle than any OS level protection - not only is the application limited from reading/writing outside it's address space (the system heap + application heap), but it's also limited from reading/writing any of the bytes between allocated blocks in the system/application heap. If the application being debugged is a C/C++ application, then Spotlight also protects the bytes between blocks in the C heap.

An additional refinement that Spotlight adds to the byte level memory protection is bounds-checking within an allocated memory block. While an out of bounds access in a block would most likely end up in the space between a block and the next, it's possible that the reference could (un)luckily end up in the next block - and not be picked up by the normal checking. The block bounds-checking feature of Spotlight means that Spotlight can automatically catch most of these styles of memory accesses, allowing the you to fix the bug.

The remaining memory protection feature of Spotlight is also the most subtle. One of the most irritating bugs on the Macintosh is when you dereference an unlocked pointer, make a call to a function that moves memory, and then access the memory via the previously dereferenced pointer. 99 times out of 100, nothing will go wrong - but that remaining 1% of cases makes life as a Macintosh programmer irritating. Spotlight will automatically detect stale handle references - even if the handle hasn't moved

Memory Leak Detection

Spotlight has some additional features to help with memory leak detection. Under the debug menu are the two options Reset Leaks and Dump Leaks….

Reset Leaks flushes the current list of 'allocated' memory blocks, such that only subsequently allocated blocks will be reported as memory leaks.

Dump Leaks… outputs a list of the currently allocated blocks, or at least allocated since the last Reset Leaks or Dump Leaks… to a selected file. Note, Spotlight doesn't display the leaks in a window - this will be added in a later version of the software.

Toolbox Validation

Spotlight provides validation for nearly 400 Toolbox functions in the current version, and more will be added in the future. Since the memory protection checks that Spotlight provides are only performed on application code, Spotlight tries to validate the parameters passed to Toolbox functions, so the application can't get the Toolbox to corrupt things on the application's behalf.

The fundamental checks that Spotlight provides are as follows:

a) All handle parameters are validated to verify that they're a legal handle within the system/application heap;

b) All memory blocks passed as parameters are validated to verify that they are fully enclosed within memory blocks;

c) If a block of memory is passed to a function that can move memory, the block is verified that it isn't contained within an unlocked handle;

d) Return values are checked - this includes indirect return values, such as MemErr().

Spotlight also provides a set of checks that are specific for groups of functions, which include:

a) ControlHandles passed to the Control Manager are validated against the list of controls in the windows in the current window list;

b) Functions that only work with Resource Manager handles or otherwise test the ownership of the handle;

c) Drawing without having a selected GrafPort (or equivalent);

d) Drawing to an unlocked GWorld;

e) Calling functions that only work on GrafPtrs with CGrafPtrs and vice-versa;

f) WindowPtrs used in the Window Manager are validated against the current window list.

Spotlight also performs other operations, such as automatically initializing newly allocated blocks with 0xF1 (other than calls like NewPtrClear) - to maximize the possibility of finding bugs.

The are also many individual checks performed on a function by function basis - too many to list here.

4 Using Spotlight

To use Spotlight you'll need to have a development environment that builds PowerPC applications and .SYM (or .xSYM for CodeWarrior users) files that contain debugger information. To run Spotlight over an application, either:

a) drag the .SYM (or .xSYM for CodeWarrior users) file onto the Spotlight application;
   or
b) start the Spotlight application and select the .SYM (or .xSYM) file from the open dialog that Spotlight displays.

Spotlight will create a specially patched version of your application (creating a temporary copy called 'ImageBackup') which it will then run, albeit with a performance penalty, with Spotlight only taking control if the application code tries to do something that is potentially dangerous. (See the In Case of Error section for a description of what happens when Spotlight detects an error). Note, the performance penalty differs from application to application, mainly depending on the balance between toolbox calls and processing that your application performs - similar in many ways to running a 680x0 application under the emulator.

Preference Settings

Spotlight instruments your application and performs tests indicated as on or off in the Preferences dialog, accessible from the 'Edit' menu, 'Preferences' menu item. Figure 2 below shows the preferences dialog.

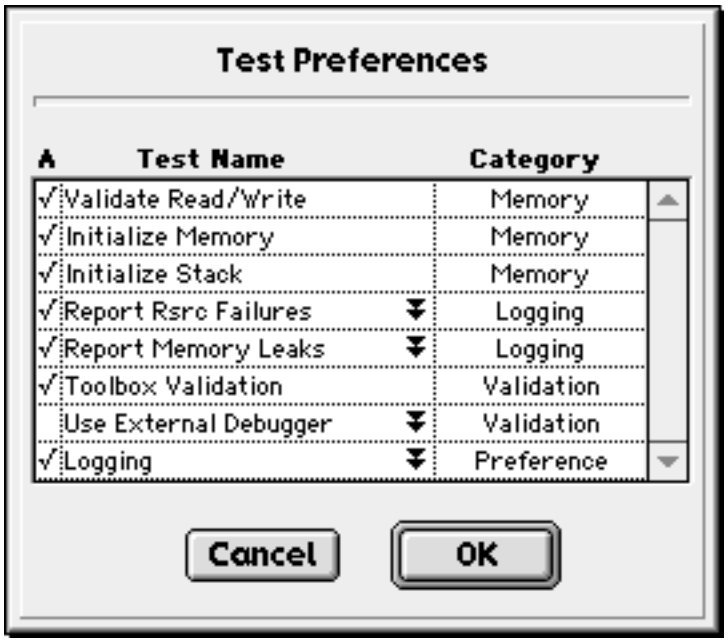

Figure 2

You activate or deactivate a particular test by clicking in the check mark column (left column) next to each preference title. You may notice that some of the preference items have a ⏬ symbol next to the preference title. This symbol indicates that if you double click on the name of that preference settings, additional options will displayed in additional dialogs allowing you to fine tune that particular preference.

For example, double clicking on the 'Logging' preference title will display the dialog in figure 3.

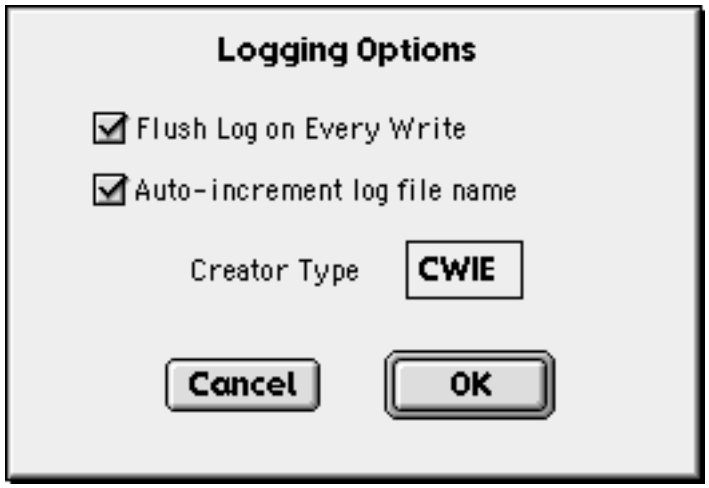

Figure 3

This dialog allows you to customize the logging preferences Spotlight will use to output testing results.

IMPORTANT
Editing of preferences can only be done before you have opened an xSYM file and instrumented some code.

Reporting Errors

During the execution of your application, Spotlight will be performing the tests activated in the Preferences dialog. If Spotlight detects something to warn the user about or detects an error, the window in figure 4 will be displayed to the user.

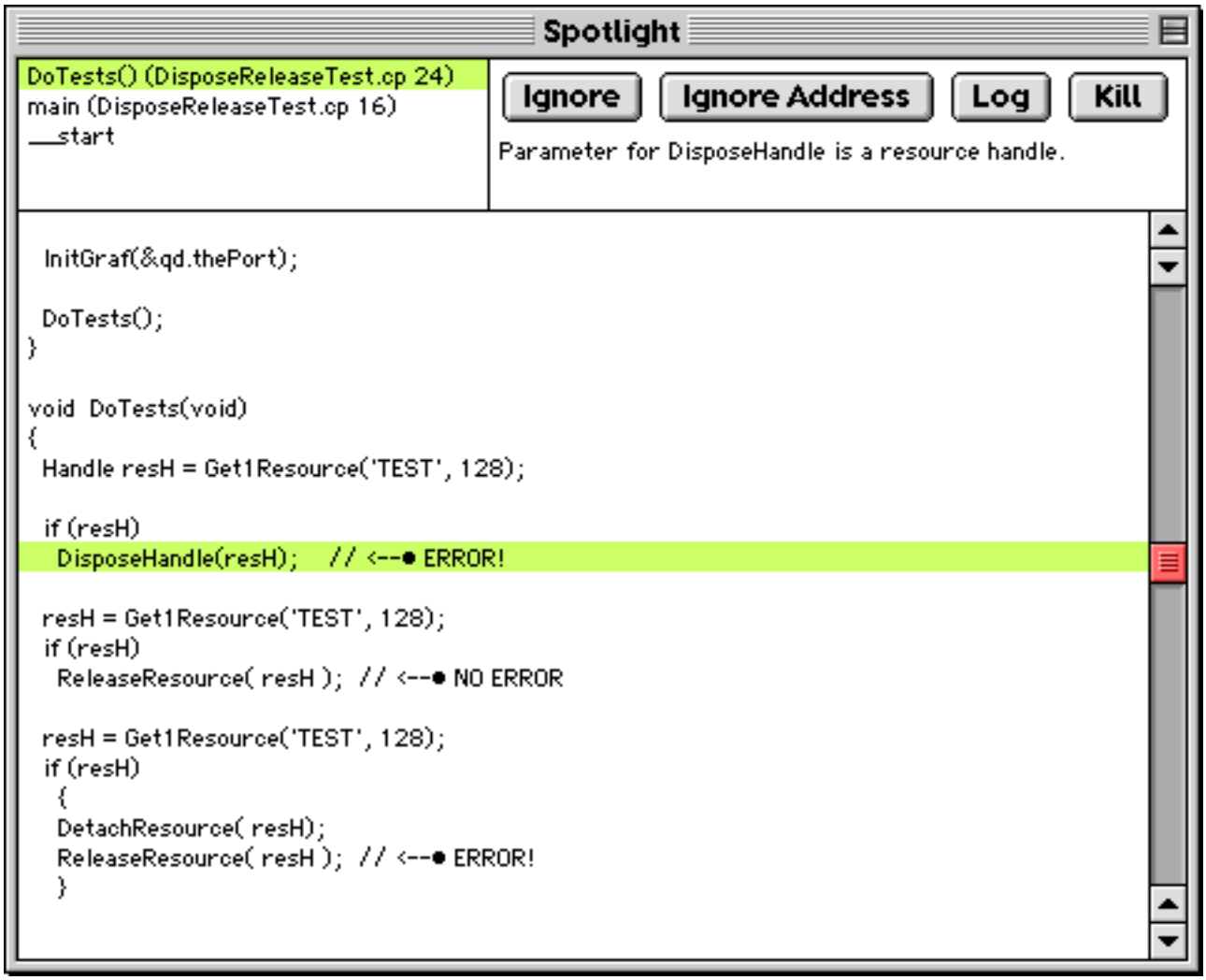

Figure 4

This areas of this window are as follows:

- The error/warning Spotlight is reporting to the user.
- A stack crawl of routines leading up to the current location.
- The source code of the offending instructions the error or warning is about.
- 4 buttons allowing control of Spotlight execution.

The four buttons and their functionality are as follows:

Ignore	Ignore the current error and continue execution. This is similar to 'G'o in other debuggers.
Ignore Address	This button saves the current PC address and indicates to Spotlight that you want to ignore any subsequent errors detected at this address.
Log	Log the current error, stack crawl, and source info in the output log.
Kill	Kill/terminate execution of the application being tested.

Once you have tested your application and quit, Spotlight will pop up with a display of all of the errors that were logged, as well as any memory that was leaked by your application. The memory leak display is broken up into:

- Leaked Handles
- Leaked Ptrs
- Leaked malloc blocks
- Leaked C++ objects
- Leaked resources

The log includes the size of each block, as well as a symbolic stack trace at the time the block was allocated. Also see the Memory Leak Detection for further memory leak detection options.

The log is automatically saved into a file called 'Spotlight Log' in the same folder as the application in case you wish to refer to it later.

5 External Debugging

Typical debugging with Spotlight will involve using Spotlight as a mini-debugger to launch, catch errors, log them or ignore them, continue execution or kill the process. Spotlight allows the user to debug their application with an external debugger like MW Debug, Jasik, or even Macsbug. If you turn on (e.g put a check mark next to that test in the options dialog) "Use External Debugger" then Spotlight will not use Power Mac DebugServices to control and debug your application. In current betas of Spotlight this means that you will need to follow some important steps to test your code. These are listed below in the section titled "***How do I use an External Debugger?***".

Why would I want to turn on "Use External Debugger"?

Using an external debugger allows you to test and debug shared libraries and plugin code resources. Previous versions of Spotlight were limited to debugging application code only. By leaving the debugging control to an external debugger (like MW Debug), Spotlight can simply instrument the executable to be tested and leave execution control up to a debugger that can handle shared libraries and code resources/plugins.

Another reason why you would want to turn this option on is so you are working in the debugging environment of choice, complete with variable and stack displays, source code displays in the fonts you want, and the ability to set break points, step, and otherwise control execution.

How do I use an external debugger?

In versions of Spotlight that support the 'Use External Debugger' option (v1.0b19e4 or later) , the following steps must be followed in order to instrument and test code with an external debugger. For the purposes of this document we will use Metrowerks' MW Debug as our example debugger.

Summary of External Debugging with Spotlight

1. Add a call to SLInit() somewhere in your PPC code initialization routine (i.e. replace __start). This can be done by using one of the supplied calls in the SpotlightAPI.h file. Specifically, SLInitialize() or SLStart(). See details below.

2. Open the xSYM file of the executable with Spotlight.

3. Launch MW Debug and open the xSYM file that you just processed with Spotlight.

4. Run and debug your code as usual. During execution, MW Debug may catch and display Spotlight detected errors.

5. Quit (or kill) the application/code you are debugging.

6. Quit the Spotlight application running in the background.

Necessary steps to using an external debugger.

1. Add a call to SLInit() somewhere in your PPC fragments initialization or startup. Because allocations can occur before your main() is ever called, you need to make sure SLInit is one of the first things called.

   If you PowerPC executable is an application, you can replace the ‘start’ PEF entry point to be a routine that calls SLInit() and then call the regular '__start' or whatever your regular entry point is.

   A typical example of replacing '__start' might be the following. In your pef linker options, replace '__start' with 'SLStart'. This routine can be found in the SpotlightAPI.c file.

```
pascal void SLStart(void)
{
      SLInit();
      __start();
}
```

   If your PowerPC executable is a code resource or shared library, you will need to call SLInit() from your ‘initialize’ PEF entry point. If you replace ‘__initialize’ then you can use the SLInitialize() routine shown below.

```
pascal OSErr SLInitialize(CFragInitBlock *theInitBlock)
{
      SLInit();
      return( __initialize(theInitBlock) );
}
```

   Note that as always, having this in your code will simple do nothing when you are not testing with Spotlight. We still recommend that you put this in a debug build though so nothing extra is left in your release builds.

2. Next, open the xSYM file of the executable you wish to test with Spotlight. This will let Spotlight instrument the binary to be tested. Normally after this step is completed, Spotlight would launch the application. Because you may be testing a code resource or shared library, Spotlight does not attempt to launch the binary leaving that up to you.

3. Launch MW Debug with the same xSYM file. Launch the application from the debugger as you would normally do. At this point you will be debugging a binary that has been instrumented by Spotlight.

4. Run and debug your code as usual. During execution, MW Debug may catch and display Spotlight detected errors. An example of what this will look like is shown in figure 5. You will notice at the time MW Debug caught the error, the stack display is showing unknown code. This is Spotlight code. Back up the stack a few calls will be routines from your application.

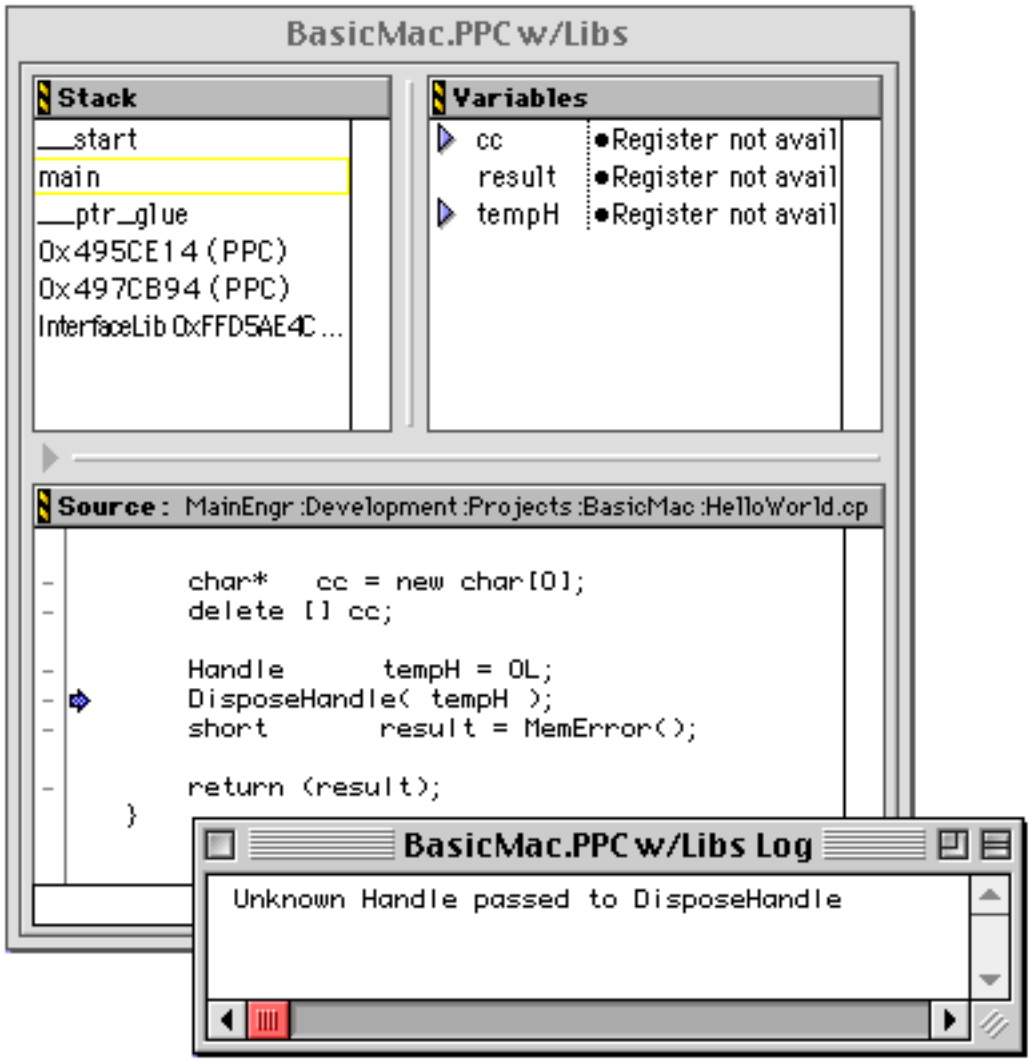

Figure 5

In the example shown in figure 5, 'main' is shown back up the stack. Click on that stack frame and you will see the variables and information for that stack frame. It's a good idea to set a breakpoint immediately after the line being executed so that when you continue execution you will break back in that stack frame; able to view and modify variables and re-execute the offending line if desired. Figure 6 shows an example of this.

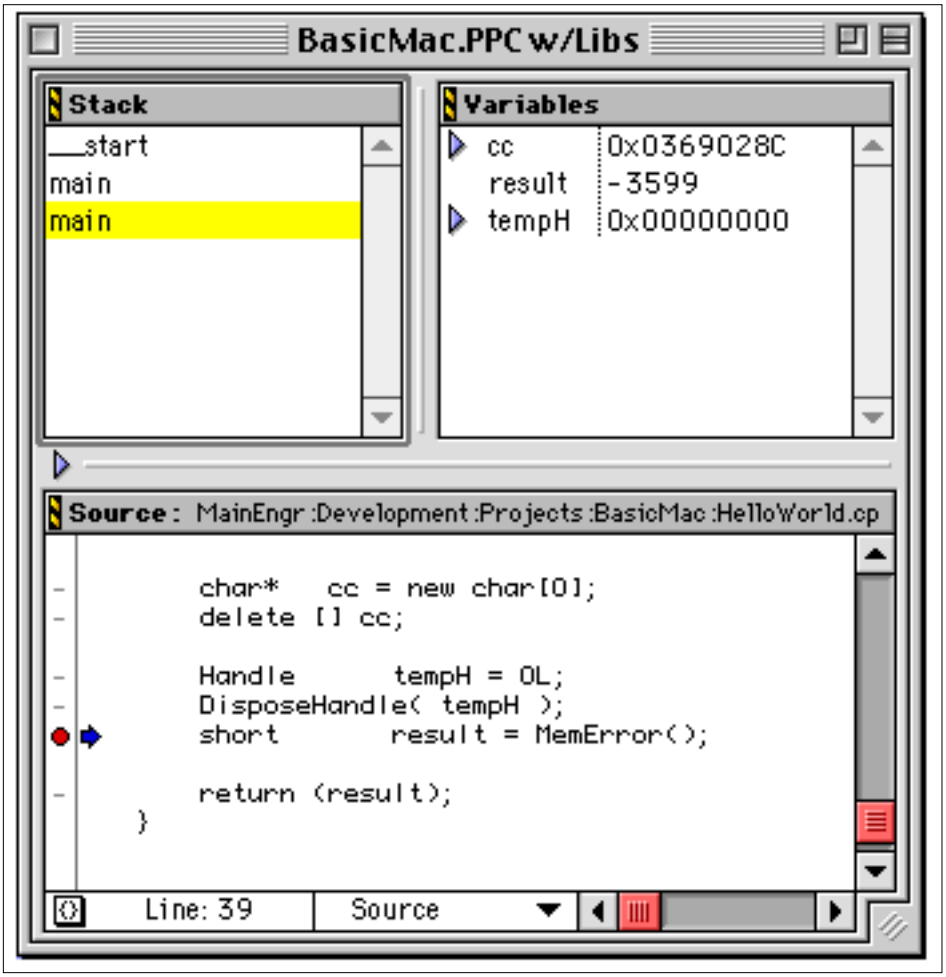

Figure 6

5. Quit (or kill) the application/code you are debugging.

6. Quit the Spotlight application running in the background. Upon quitting Spotlight, the output log will be generated. This includes the leaks log and reported errors.

Things to watch out for!

1. Do not forget to quit Spotlight after each run of your application.
2. Do not restart a program or code you have just debugged with MW Debug by hitting CMD-R (run) without first quitting Spotlight and restarting it.

6 Spotlight API

Included as part of Spotlight are interface files to the SpotlightAPI which contains several functions for controlling Spotlight.

SLDisable and SLEnable

These functions are used to disable and enable Spotlight validation and you can use these functions around any pieces of code that perform operations that Spotlight incorrectly assumes are 'bad'.

SLEnterInterrupt and SLLeaveInterrupt

These functions are used to notify that the bracketed code is interrupt level code and that (a) Spotlight should detect if an application is trying to call interrupt unsafe functions, and (b) Spotlight should defer notification of any problem until the application is no longer running at interrupt level. Certain interrupt handlers are automatically caught by Spotlight, so it's possible you won't need to use these functions. Currently, Spotlight only detects sound completion routines, but more will be added.

Also note that all of these functions are completely harmless when you aren't running your application under Spotlight, and can be left inside a shipping application (unless they are in a really speed critical section).

Customer Support

Onyx Sales and Product Information

If you have a product or sales related question, please call (941) 795-7801. Our customer sales representatives will be able to give you information about Onyx products or have information sent to you. Requests for information can also be faxed to (941) 795-5901.

Onyx Technical Support

Technical support for Spotlight is primarily through electronic mail services. We have a company support forum on America Online in the Developer Forum. Look in our company support forum for updates, patches, and other additions to Spotlight. This is an excellent place to post questions relating to Spotlight.

We can be reached at the email addresses below.

Internet: support@onyx-tech.com

Please contact us at one of these addresses if you have any questions or problems using Spotlight. Our average response time is 12 hours.

Customer Suggestions and Feedback

We are always looking for ways to improve Spotlight and make it better serve your developmental needs. If you have suggestions for improvements to Spotlight, please feel free to send them to us at one of the addresses above.

Spotlight Web Page

The Onyx Technology Spotlight Worldwide Web page is available at the following location.

http://www.onyx-tech.com/

The web site contains Spotlight information, demos, sample code, updaters for various versions, press releases, articles written about Spotlight, as well as links so related web sites.

Spotlight FTP address

The Onyx Technology ftp site can be found at the following location.

ftp://ftp.onyx-tech.com/pub/Spotlight/

This directory contains Spotlight related materials and is the destination for FTP links from the above mentioned web page.