Debugging WebObjects

by Theresa Ray of Tensor Information Systems, Inc.

Sponsored by Apple Computer, Inc.
Apple Developer Connection

Debugging WebObjects

by Theresa Ray of Tensor Information Systems, Inc.

A quick reference guide to the major topics involved in debugging a WebObjects application including ProjectBuilder's debugging tools for both Objective-C and Java applications, as well as other logging and debug techniques available outside ProjectBuilder.

Developing a robust application for either a web or client-server interface is critical for your business, particularly with web applications, which are generally developed for use by people outside your company. A web site makes an impression on its users, in many instances defining the way they feel about your company. Therefore, developing a robust application is critical to your business, and one of the best ways to solidify an application is to test and debug it thoroughly. A good test plan is as important to a solid system as a good architecture. But once your test plan identifies problems with the application, the source of that problem is not always obvious. Debugging tools are critical to this phase of the project, and are also extremely useful during the initial development phase itself.

ProjectBuilder's Debugging Tools

ProjectBuilder, Apple's application development tool, includes one or more built-in debuggers, depending on the version of WebObjects you are working with. All versions of WebObjects' ProjectBuilder include an extended version of the GNU debugger from the Free Software Foundation. The version of the GNU debugger supplied with ProjectBuilder is usually referred to as gdb and has been extended to support the use of Objective-C code, while still remaining fully compatible with the original version.

WebObjects 3.5 included a Java debugger, jdb, available from the command-line. But in WebObjects 4.0, jdb has been integrated with ProjectBuilder for debugging Yellow Box Java applications, and uses a customized subset of jdb commands. Project Builder integrates the Java Debugger and gdb so that, with projects that consist of Java code and other code (Objective-C, C, or C + +), you can use both debuggers in the same session, switching between them as necessary. Currently, there is no mixed-stack backtrace. Both gdb and the Java Debugger, like most debuggers, enable you to perform the following tasks:

- Start the program with or without command-line arguments
- Stop the program when a specific condition occurs
- Examine registers and the stack when the program has stopped at a user-specified condition
- Change some behavior of the application on the fly in order to determine the correct solution without the need for recompiling.

Building an Application for Debug

In order to debug your application, first you must compile your application so that it contains the proper debugging symbols. In ProjectBuilder, go to the Build Options panel. Click on the Options button (the check mark), and choose "debug" as the target. If your previous build wasn't with the debug target, click the Make Clean button (the broom) in the Build Options panel. When that is complete, click the Build button (the hammer). You now have the appropriate executable for debugging your application and you are ready to go!

Running the Debugger

In WebObjects 4.0, ProjectBuilder knows which debugger to use – gdb or the Java Debugger – based on the source code type of your application. The steps involved in accessing the debugger are the basically the same, regardless of whether you have a purely Objective-C based application or a mixed Objective-C and Java application. But you should check the Debug Options (click on the checkmark icon) in the Launch panel and make sure that the appropriate debuggers are enabled (there are two checkboxes – one to enable gdb and one to enable the Java Debugger).

To start the debugger, click on the Debug icon (the spray can) in the Launch panel. The appropriate debugger starts (gdb, the Java Debugger, or both). After gdb's initial startup, the debugger pauses to allow you to set breakpoints in your application (just double-click on the gray band to the left of the line of code you wish to break on, and a pointer will appear indicating that the breakpoint has been set). Because the Java VM is an interpreter, you do not need to suspend the Java Debugger to set breakpoints (and by default it does not pause), but you can click the "Suspend" button if you wish to prevent the application from starting before setting your breakpoints.

Setting Breakpoints and Stepping Through Code

When the application reaches a preset breakpoint, the application pauses, and the line of code is highlighted in yellow with a red pointer in the left gray margin to indicate the next step to be executed by the application. At this point, you may examine the stack (click the Task Inspector – the gear – in the Launch Panel to display breakpoints and the stack), print register or variable values, step into and over code, etc.

To step into or over code, click the appropriate icon in the Launch panel. The Step Over icon allows you to execute the line of code and move on to the next line of code, and is identified by an arrow pointing over a pair of parentheses. The Step Into icon allows you to step through any methods executed by the current line of code, and is identified by an arrow pointing between two parentheses.

Examining Objects and Variables

To print or examine objects or variables, highlight the object, variable or expression to inspect (after the statement that assigns its value) and then click on the appropriate icon in the Launch panel. There are three icons for inspecting the value of an object or variable. One icon is the Print Value icon, which prints the value of a variable and is indicated by a panel with a red arrow. Another icon is the Print Reference icon, which prints the value referenced by a variable and is indicated by a panel with a red arrow and an asterisk (this is useful for printing the instance variables of a class or object). The last is the Print Object icon, which displays the object's self-description method by calling the object's toString method and is indicated by a panel with a red arrow and a cube.

At the debugger's command line, you can also type the following to get more information:

`info args` – shows you information about the current stack frame; self, _cmd, arguments, etc.

`info classes` – shows all loaded classes

`p *(id *)($fp + x)` – depending on x you get self, _cmd, the calling frame, arguments, etc.

Online Documentation

Both gdb and the Java Debugger allow many more options for debugging than a document of this magnitude can cover. The online documents contain a detailed reference for gdb (and in WebObjects 4.0, for the Java Debugger as well). You can find the online documents for the debuggers as follows:

In WebObjects 3.x, launch the OpenStep Books Online and select the link for gdb, the GNU debugger.

In WebObjects 4.0, launch the WebObjects Info Center. The gdb documentation is found by clicking on Books, then Development Environment, then Debugger Reference: The GNU Source-level Debugger. The Java Debugger documentation is found by entering "jdb" into the search text field and clicking search.

Other Debugging and Logging Techniques

Debugging a WebObjects application can often require more than just running the built-in debugger. Sometimes a different approach is required, as for scripted web components for which the source-level debuggers are not useful, or for a released, live, production application that only intermittently has an error. Tracking down the exact path to failure can be very difficult, and setting breakpoints for stepping through the application may be neither practical nor useful in determining the problem.

Logging and Redirecting Output to a File

Logging information at key points in your application using NSLog's methods or using logWithFormat: for scripted components can often provide insight as to the behavior of the application just before an exception. This can be particularly useful for a live production application that only intermittently produces an error due to a user running a specific query or clicking on a single broken link. To ensure that you trap all the output the application sends to STDOUT and STDERR, you can add the following to your application startup command and redirect all output to a file called command.log:

```
< command name > 1 > command.log 2 > &1
```

If the terminal you used to launch the command closes, or you need someone else to launch the command, this extension guarantees that you will have access to all output from this application regardless.

Catching and Handling Exceptions

Another technique that is very useful for production applications is to implement methods in the Application subclass like handleSessionCreationErrorInContext:, handleSessionRestorationErrorInContext:, handlePageRestorationErrorInContext:, and handleException:inContext:. Within these methods, you can log information critical to the exception for later review and return a WOResponse that shows a user-friendly (as opposed to the default developer-friendly) error message. Another method that provides similar functionality is NSSetUncaughtExceptionHandler() which lets you add more details to the output for an uncaught exception.

For even more exception handling capability, add the following method to your xxx_main.m. This trick lets you get more information into the infamous "freed(id): message sent to freed object" message that occurs when your Objective-C memory management goes awry:

```
void myError (id anObject, const char *format, valist ap )
{
     char buf[BUFSIZ];
```

```
     vsprintf( buf, format, ap ) ;

     printf( "[%d][T 0x%x] %s\n", (int) getpid(),
      (unsigned) [ NSThread currentThread ] , buf ) ;
}

int main ( int argc, const char *argv[] )
{
     //-- Swap the default error printing function.

     _defaultError = _error ;
     _error = _myError ;

     // … do all your fun stuff here as usual to run the
     // application
}
```

Logging SQL

Use the following at the command line on any platform before running your application to log all the SQL generated by the Enterprise Object Frameworks as it communicates with the database:

```
defaults write NSGlobalDomain EOAdaptorDebugEnabled YES
```

When you want to turn the SQL logging back off, run the following command:

```
defaults write NSGlobalDomain EOAdaptorDebugEnabled NO
```

Logging NSConnection Information

If you are using Distributed Objects in your application as well, you can use the following undocumented class method to turn on and off NSConnection debug information (but be warned, it produces a LOT of output):

```
+ (void) _enableLogging:(BOOL)on ;
```

Other Tools Provided with WebObjects

ObjectAlloc, a tool provided with WebObjects, is a great tool for inspecting and debugging the allocation and deallocation of memory for your Objective-C applications, but it is only available on NT. Read the documentation provided with WebObjects for more information on this tool.

WebObjects 4.0 provides a whole host of information available through the Monitor application and other new debug and statistics features (for example, there are new features to enable WOComponents to log debug information) that are too extensive to cover here. Refer to the WebObjects 4.0 documentation for more information.

Conclusion

These tools and techniques should provide you with a great start for debugging your pre and post production WebObjects application. As stated before, gdb and the Java Debugger are invaluable tools for examining in detail the behavior of your application, and you should quickly become familiar with the capabilities of these debuggers.

Print out the on-line documentation, read it, and play with the debugger on the examples provided with WebObjects until you are comfortable with its operation. And hopefully the other techniques presented in this document will supplement whatever insight the debuggers cannot provide.

Resources…

http://gemma.apple.com/techinfo/techdocs/enterprise/WebObjects
WebObjects Developer's Guide
Enterprise Objects Framework Developer's Guide
http://www.omnigroup.com/MailArchive/WebObjects
http://www.omnigroup.com/MailArchive/eof
http://www2.stepwise.com/cgi-bin/WebObjects/Stepwise/Sites
ftp://dev.apple.com/devworld/Interactive_Media_Resources
http://www.apple.com/developer
http://developer.apple.com/media
http://enterprise.apple.com/NeXTanswers

About the Author…

Theresa Ray is a Senior Software Consultant for Tensor Information Systems in Fort Worth, TX (*http://www.tensor.com)*. She has worked as a consultant on WebObjects projects for a wide variety of clients including the U.S. Navy, the United States Postal Service, America Online, and Proctor and Gamble. Her experience spans all versions of WebObjects, from 1.0 to 4.0 beta, several versions of EOF, from 1.1 to 3.0 beta, AppKit, NEXTSTEP 3.1 to OPENSTEP 4.2, Rhapsody for Power Macintosh, and yellow-box for NT. In addition, she is an Apple-certified instructor for WebObjects courses.

Tensor Information Systems is a Apple partner providing systems integration and enterprise solutions to its customers. Tensor's employees are experienced in all Apple technologies including OPENSTEP, NEXTSTEP, Rhapsody, EOF and WebObjects. Tensor also provides Apple-certified training in WebObjects, Oracle consulting and training, as well as systems integration consulting on HP-UX.

You may reach Theresa by e-mail: *theresa@tensor.com*