

SourceBug Reference

For SourceBug version 1.0

Apple Computer, Inc.
© 1991, Apple Computer, Inc.
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, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc. Printed in the United States of America.

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014-6299
408-996-1010

Apple, the Apple logo, APDA, AppleLink, A/UX, LaserWriter, MacApp, Macintosh, MPW, MultiFinder, and SADE are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Finder and SourceBug are trademarks of Apple Computer, Inc.

Illustrator and PostScript are registered trademarks of Adobe Systems Incorporated.

FrameMaker is a registered trademark of Frame Technology Corporation.

Frutiger is trademark of Linotype AG and/or its subsidiaries.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

Varietyper is a registered trademark of Varietyper, Inc.

Simultaneously published in the United States and Canada.

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, **APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Figures vii

Preface **About This Manual** ix

What's in this manual? ix
Aids to understanding x
 For more information x

Chapter 1 **Introduction to SourceBug** 1

What is SourceBug? 2
Hardware and software requirements 2
Installing SourceBug 3
About SourceBug 3
 SourceBug's capabilities 3
 SourceBug and MacsBug 4
 SourceBug and SADE 5
 SourceBug and the MacApp debugger 5

Chapter 2 **Using SourceBug** 7

Before you begin 8
Launching SourceBug 9
Using the Browser window 10
 Displaying classes and methods 11
 Using the code listing 12
 Detecting inheritance 12
 Selecting text in a code listing 13
 Cloning code panes 14
 Setting breakpoints 16
Using the Stack Crawl window 17
 Selecting methods (activations) on the stack 19
 Controlling program execution 21
 Animation mode 23
 Using variable windows 23
Using the Inspector window 25
 Displaying the fields of an object 26

The File menu	30	
Open... (Command-O)	30	
Close (Command-W)	31	
Save Log As...	32	
Quit (Command-Q)	32	
The Edit menu	32	
Undo/Redo (Command-Z)	33	
Standard Macintosh editing commands	33	
Show Registers	33	
Show FPU Registers	34	
Show Log Window	34	
The Class menu	35	
Alphabetical Classes	36	
Hierarchical Classes	36	
The Member menu	36	
View Source	36	
View Assembler	36	
Find Code For... (Command-F)	37	
Find Code for “ ” (Command-H)	37	
Find Inherited “ ”	37	
Find Implementations of “ ”	37	
Source File for “ ”	37	
The Control menu	38	
Run (Command-R)	38	
Kill (Command-K)	38	
Switch to Low-level Debugger	38	
Step (Command-S)	39	
Step Into (Command-I)	39	
Step Out (Command-P)	39	
Animate	39	
Set Breakpoint at Failure	39	
Clear All Breakpoints	39	
The Inspect menu	40	
New Inspector Window	40	
Evaluate...	42	
Evaluate “ ” (Command-E)	42	
Evaluate Self	43	
View as	43	
The Windows menu	43	

Using Manual Breakpoints With SourceBug 45

Putting breakpoints into your source code manually	45
Break and print statements using SysBreak	46
Print statements using printf	46

Index 49

Figures

Chapter 2

Using SourceBug 7

Figure 2-1	Choosing an application to debug	9
Figure 2-2	“Nothing” Browser window	10
Figure 2-3	Classes of “nothing” and methods of TApplication	11
Figure 2-4	Code listing	12
Figure 2-5	Implementations of DoMenuCommand	13
Figure 2-6	Cloned code listing	15
Figure 2-7	Breakpoint diamond icon	16
Figure 2-8	“Nothing” application display	17
Figure 2-9	Stack Crawl window	18
Figure 2-10	Source code for stack activation	20
Figure 2-11	Source and disassembly after stepping	22
Figure 2-12	The savedPort variable	24
Figure 2-13	Inspector window	25
Figure 2-14	Inspector variable window	26

Chapter 3

Menu Reference 29

Figure 3-1	File menu	30
Figure 3-2	Browser window for the “nothing” application	31
Figure 3-3	Edit menu	32
Figure 3-4	Registers for the “nothing” application	33
Figure 3-5	Sample FPU registers	34
Figure 3-6	Sample Log window	35
Figure 3-7	Class menu	35
Figure 3-8	Member menu	36
Figure 3-9	Control menu	38
Figure 3-10	Inspect menu	40
Figure 3-11	A new Inspector window	41
Figure 3-12	An Inspector window with a class selected	41
Figure 3-13	Inspector window showing fields	42

About This Manual

SourceBug is a source-level interactive debugger that you can use with Macintosh programs written in assembly language or in languages such as Pascal, C, or Fortran. SourceBug is also effective with programs written in Object Pascal or C++, with or without the MacApp application framework. It can be used as either a source-level debugger or an assembly-level debugger, no matter what language the target program is written in.

What's in this manual?

This manual describes the most important components and features of SourceBug, in both reference and tutorial form.

- Chapter 1, “Introduction to SourceBug,” provides an overview of SourceBug. It describes the hardware and software configurations you need for using SourceBug, explains how to install SourceBug, and discusses the differences between SourceBug and other debuggers.
- Chapter 2, “Using SourceBug,” helps you get started with SourceBug and describes the basic techniques for using SourceBug and the characteristics of its different windows.
- Chapter 3, “Menu Reference,” describes the SourceBug menus and explains in detail the menu commands that are unique to SourceBug.
- The Appendix, “Using Manual Breakpoints With SourceBug,” describes how to manually put break and print statements into your application's source code for use with SourceBug.

Other useful references include the Macintosh Programmer's Workshop Development Environment manuals and *MacApp Tools and Languages*.

Aids to understanding

Look for these visual cues throughout the manual:

▲ **WARNING**

Warnings like this indicate potential problems. ▲

IMPORTANT

Text set off in this manner presents important information. ▲

NOTE

Text set off in this manner presents notes, reminders, and hints. ◆

For more information

APDA (Apple Programmers and Developers Association) offers worldwide access to a broad range of programming products, resources, and information for anyone developing on Apple platforms. You'll find the most current versions of Apple and third-party development tools, debuggers, compilers, languages, and technical references for all Apple platforms. To establish an APDA account, obtain additional ordering information, or find out about site licensing and developer training programs, please contact

APDA

Apple Computer, Inc.
20525 Mariani Avenue, M/S 33-G
Cupertino, CA 95014-6299

800-282-2732 (United States)

800-637-0029 (Canada)

408-562-3910 (International)

Fax: 408-562-3971

Telex: 171-576

AppleLink address: APDA

If you provide commercial products and services, please call 408-974-4897 for information on the developer support programs available from Apple.

C H A P T E R 1

Introduction to SourceBug

This chapter introduces some fundamental debugging concepts and describes how SourceBug can aid you with the debugging process. It describes the hardware and software you need and explains how to install SourceBug. It also explains how SourceBug differs from other debuggers that are available for debugging Macintosh applications and describes the situations for which it is best suited.

What is SourceBug?

SourceBug is a debugger—that is, a program that helps you detect and correct errors in other programs. You can use SourceBug to debug applications written in any language supported by the Macintosh Programmer’s Workshop (MPW), including Pascal, C, Fortran, assembly language, Object Pascal, and C++. You can also use SourceBug to debug applications that use the MacApp application framework.

Hardware and software requirements

To run SourceBug, you need the following hardware and software:

- Macintosh Plus computer, a Macintosh SE computer, any computer from the Macintosh II family, or a Macintosh computer that uses future members of the MC68000 microprocessor family. A Macintosh II computer or later model is recommended.
- Macintosh system software version 7.0 or later or Macintosh system software version 6.0.5 or later with MultiFinder version 6.1B9.
- 2.5 MB RAM for SourceBug in addition to that required to run your application.
- MPW version 3.2 or later.
- MacApp version 2.0 or later (if you are debugging a program built with MacApp).
- A/UX version 2.0 or later (if you are running under A/UX).

Installing SourceBug

To install SourceBug, simply drag the SourceBug folder from the release disk to any location on your hard disk. The SourceBug folder does not need to be associated with the MPW folder, although you can place it in the MPW folder if you wish.

IMPORTANT

If you are using version 6.0.x of the Macintosh system software, you must also replace the MultiFinder program that is in your System Folder with MultiFinder version 6.1B9, which is on the SourceBug release disk. The MultiFinder program on the SourceBug release disk contains special code for controlling and accessing the application state. This additional code supports debugging but does not affect your normal use of MultiFinder. To install the special version of MultiFinder, move your current MultiFinder file out of the System Folder, copy the new version in, and restart your computer. If you already have MultiFinder version 6.1B9, or if you are running system software version 7.0 or later, skip this step. ▲

The Samples folder inside the SourceBug folder contains the sample applications and source files that you need for working through the examples in Chapter 2, “Quick Start,” and Chapter 3, “Using SourceBug.” You can place this folder anywhere.

About SourceBug

This section describes SourceBug’s capabilities and compares it to the other debuggers available from Apple Computer, Inc.

SourceBug’s capabilities

SourceBug provides basic debugging operations, such as setting breakpoints, controlling program execution, and displaying the contents of variables. You can use SourceBug to debug any application written in the MPW environment. As the name implies, SourceBug is a source-level debugger; it allows you to debug your programs in the language in which they were written or, if you wish, in assembly language. In contrast, a low-level debugger such as MacsBug displays every program in machine language, no matter what language the program was originally written in. Thus, you have to understand 68000 machine language to use MacsBug, but you don’t have to understand assembly language to use SourceBug.

SourceBug also provides special support for debugging MacApp and object-oriented code. For example, SourceBug offers a browser interface to display the classes and methods of a MacApp or object-oriented program, and it provides an inspector interface—closely modeled on the MacApp Inspector—to display the contents of the fields of instantiated objects.

NOTE

SourceBug displays only Object Pascal classes and classes in C++ that are descendants of the Pascal Object class. It does not display C++ classes that are not descendants of Pascal Object. All classes in MacApp are descendants of Pascal Object. ♦

SourceBug takes full advantage of the Macintosh User Interface, providing an easy-to-use window and menu interface. You can accomplish most SourceBug tasks with the mouse, by selecting either text in a window, names from a list, or commands on a

menu. SourceBug displays information in separate windows that you can place anywhere for your convenience.

SourceBug and MacsBug

MacsBug is a low-level debugger that displays a program in machine instructions no matter what language the program was originally written in.

You can use MacsBug in conjunction with SourceBug. A SourceBug menu command, Switch to Low-level Debugger, calls MacsBug, making it easy to use both programs to debug an application. Although SourceBug provides some machine-level debugging, MacsBug provides many other features, such as heap check and scramble, memory dump capabilities, and the ability to step through ROM or other system code that SourceBug doesn't.

In addition, you can run MacsBug in cases in which you cannot use SourceBug; for example:

- If RAM is so severely limited that you are unable to run SourceBug, you can probably run MacsBug because it takes up very little space in memory.
- In a severe system crash, SourceBug may not operate because it uses the system software extensively. MacsBug, on the other hand, makes little or no use of the system software and should still be available.

SourceBug and SADE

The SADE program, like SourceBug, is a source-level debugger and provides basic functions such as setting breakpoints, controlling program execution, and examining the contents of variables. In addition, SADE has a powerful command language that allows you to do things that you can't do with SourceBug, such as

- changing the content of variables and then executing the program to see the effect
- writing and running scripts to automate debugging tasks
- debugging MPW tools
- setting breakpoints on traps
- setting conditional breakpoints

SADE does provide support for MacApp and object-oriented code, but it lacks the browser and inspector interfaces that make this code readily accessible in SourceBug.

SourceBug and the MacApp debugger

In MacApp versions 3.0 and later, most of the powerful debugging features such as the Inspector have been removed, because SourceBug performs the same functions with an easier implementation (for example, SourceBug reads the symbol file, while MacApp requires that application writers put fields methods into their source code).

MacApp provides basic error checking when you compile with debugging turned on. SourceBug does not have this feature.

Using SourceBug

Using SourceBug

This chapter shows you how to use SourceBug. It explains the different windows and panes in which SourceBug displays information, and it describes basic techniques for debugging, such as setting breakpoints, stepping through the program, and examining variables.

Before you begin

You can use SourceBug to debug any application written with MPW. However, keep these points in mind:

- The application must compile and link successfully or you cannot use SourceBug to debug it.
- The application must have an associated symbol file (*appname.sym*) generated by the MABuild command's `-sym` option (for a MacApp application) or by the Compile and Link commands' `-sym on` option (for non-MacApp applications). The symbol file must be located in the same directory as the application.

Although this chapter is not a step-by-step tutorial, it does use many examples, so you may find it helpful to run SourceBug as you read the chapter. You may find it helpful to select “nothing” (in the MacApp Examples folder) as the application to debug, because “nothing” has been successfully compiled and linked, has a proper symbol file, and is referred to in this chapter's examples.

NOTE

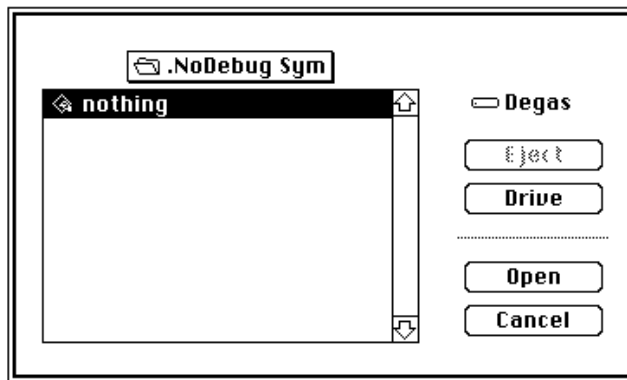
The examples in this chapter were built with MacApp version 3.0. If you use a different version of MacApp, you may not see exactly the same displays as those that appear here. The differences are minor enough that they shouldn't cause you any trouble in following the examples. For example, in Figure 2-4, the method name that appears is `DoCommandKeyEvent`, but under MacApp version 2.0 it would be `DoCommandKey`. ♦

If you are not using MacApp, you can use one of the sample MPW applications, or, if you have an application of your own that compiles and links successfully and has a proper symbol file, you can use it as the target as you read through this chapter.

Launching SourceBug

Launch SourceBug as you would any other Macintosh application, by double-clicking its icon in the Finder. SourceBug displays a Macintosh Standard File dialog box, as shown in Figure 2-1, from which you can select an application to debug (the target application).

Figure 2-1 Choosing an application to debug



Browse through the dialog box and select an application (the “nothing” application is recommended).

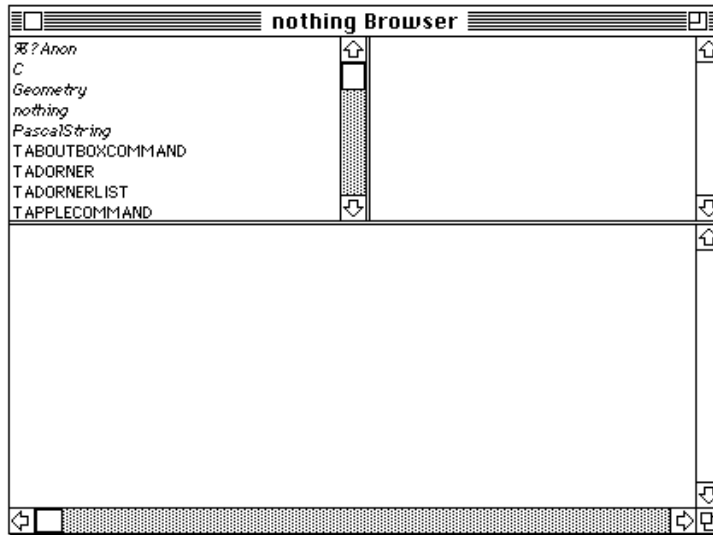
NOTE

You can run SourceBug without a target application if you wish by choosing Cancel from the dialog box. Later on you can specify a target application with the Open command in the File menu. ♦

Using the Browser window

When you have selected an application to debug, a Browser window appears. Figure 2-2 shows the Browser window for the “nothing” application.

Figure 2-2 “Nothing” Browser window



As you can see, the window is divided into three panes:

- The upper-left pane lists classes.
- The upper-right pane lists methods.
- The lower pane displays code.

At this point, only the upper-left pane has information in it. Once you select a class, SourceBug displays its methods in the upper-right pane; when you select a method, SourceBug displays the code for it in the lower pane. The rest of this section explains these three panes in detail.

▲ **WARNING**

If you close the Browser window for an application, SourceBug removes that application as the target and closes any other windows that are related to the application. ▲

You can expand or contract the size of each pane in the Browser window (and in other SourceBug windows with multiple panes) relative to the other panes. To do so, place the pointer on the line separating the panes (the pointer becomes a pair of opposing arrows), and then click, drag, and release at the desired point.

Displaying classes and methods

In the upper-left pane of the Browser window, SourceBug lists the classes in the target application; however, it lists only Pascal classes (for an Object Pascal application) or descendants of the Object Pascal class (for a C++ application). It also lists dummy classes that correspond to the compilation units or source files comprising the application, and another dummy class, `%?Anon`. Classes appear as regular text, whereas source code and dummy classes appear in italics.

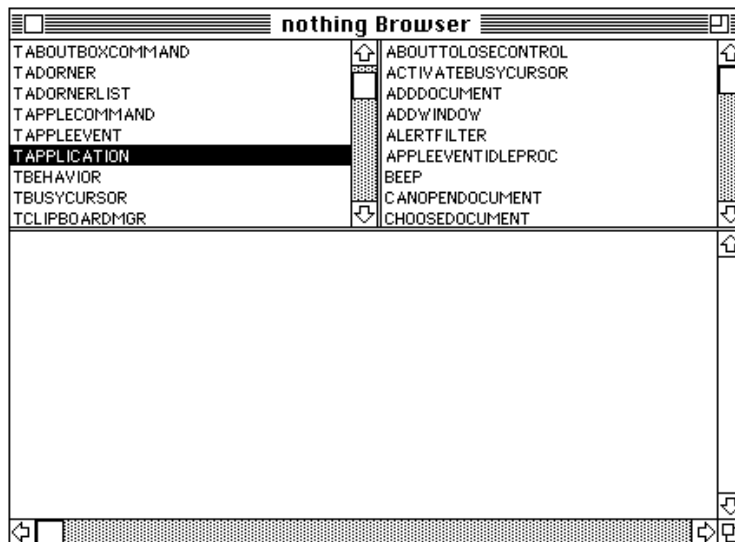
If you scroll through the upper-left pane, you can see the MacApp classes used in the “nothing” application; by convention, MacApp classes begin with the letter *T*. You should also see the `%?Anon` dummy class, the nothing source file (which contains the main program), and the MacApp source files that define the methods used in “nothing”; by convention, MacApp source files begin with the letter *U*.

If you click a class, SourceBug lists the methods that belong to the class in the upper-right pane. If you click a compilation unit or source file, the upper-right pane lists all the routines belonging to the unit that are not methods of Pascal classes; that is, it lists ordinary Pascal or C procedures and routines, C++ member functions of classes that are not descendants of Pascal Object, and library glue routines.

If you click `%?Anon`, the upper-right pane lists all routines for which there is no source code.

If you scroll in the upper-left pane and select a class such as `TApplication`, the upper-right pane lists its methods, as shown in Figure 2-3. You can scroll through the upper-right pane to see additional methods.

Figure 2-3 Classes of “nothing” and methods of `TApplication`



If the target application is written with procedural code, the upper-left pane lists compilation units and source files and the `%?Anon` dummy class only.

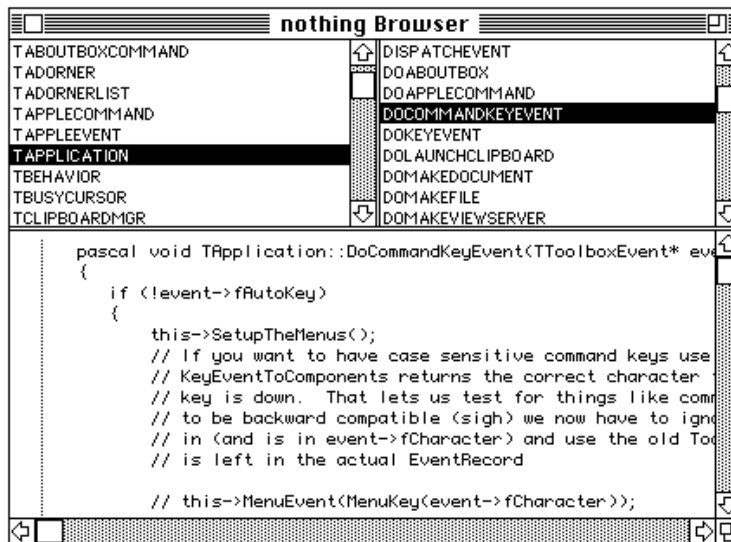
Using SourceBug

You can view the classes in alphabetical order (the default) or in class hierarchical order. Use the Alphabetical Classes and Hierarchical Classes commands in the Class menu to change the class display.

Using the code listing

When you select a class and method, SourceBug displays the source code (by default) or a disassembly of the method in the lower pane of the Browser window. For example, if you select TApplication from the list of classes and scroll down to select DoCommandEvent from the list of methods, the lower pane displays the TApplication.DoCommandEvent method, as shown in Figure 2-4.

Figure 2-4 Code listing



To see a disassembly of the code in the lower pane, use the View Assembler command in the Member menu; use the View Source command to change back to a source code display. If SourceBug cannot find source code for a particular method or routine, it automatically displays a disassembly even if you choose View Source.

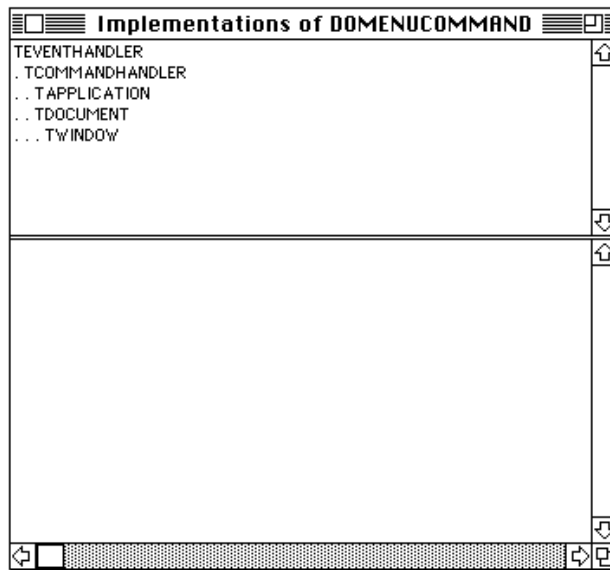
Detecting inheritance

One of the key concepts in object-oriented programming is inheritance. You can quickly find out which classes implement a particular method, and you can view the source code to compare the different implementations. To see the different implementations of a method, highlight it in the upper-right pane and choose the Find Implementations of “ ” command in the Member menu.

Using SourceBug

For example, if you choose TApplication in the list of classes and DoMenuCommand in the list of methods, choosing Find Implementations of DoMenuCommand displays a new window listing the classes in which DoMenuCommand is implemented, as shown in Figure 2-5.

Figure 2-5 Implementations of DoMenuCommand



If you select any of the classes shown in Figure 2-4, SourceBug displays in the lower pane the implementation of DoMenuCommand for that class. To see the implementation of a method in the superclass of the class you just selected, use the Find Inherited “ ” command in the Member menu. If the current implementation is at the top of the class hierarchy, Find Inherited “ ” is dimmed.

Selecting text in a code listing

When you put the pointer in the lower pane to the right of the dotted line, it appears as an I-beam, which allows you to select text. You can click and drag the I-beam to highlight a range of text, double-click to highlight a word, or triple-click to highlight one source statement (or a line of assembly language if a disassembly is being displayed).

You can use the Copy command in the Edit menu to copy text that is highlighted.

NOTE

Although SourceBug allows you to copy code, SourceBug does not have a text editor; that is, you cannot change your source code by altering it in a SourceBug window. ♦

Using SourceBug

If you want to find the source code for a particular method, you can do so by highlighting the name of the method and then using the Find Code for “ ” command in the Member menu. For example, `TApplication.DoCommandEvent`, the method displayed in Figure 2-4 earlier, calls the `SetupTheMenus` method. If you highlight `SetupTheMenus` in the lower pane and choose Find Code for `SetupTheMenus`, SourceBug displays a new window containing the code for `TApplication.SetupTheMenus`.

Cloning code panes

SourceBug enables you to have two or more views into the same source code by cloning a code pane. Hold down the Option key—which turns the pointer into a small window—click a code pane, and drag and release to create a window that lists the same code as the code pane.

With two views of the same method, you can display one as source code and the other as a disassembly. For example, Figure 2-6 shows a source view and a disassembly of the `TApplication.DoMenuCommand` method.

Using SourceBug

Figure 2-6 Cloned code listing

nothing Browser

TABOUTBOXCOMMAND	DOLAUNCHCLIPBOARD
TADORNER	DOMAKEDOCUMENT
TADORNRLIST	DOMAKEFILE
TAPPLECOMMAND	DOMAKEVIEWSERVER
TAPPLLEEVENT	DOMENUCOMMAND
TAPPLICATION	DOSETCURSOR
TBEHAVIOR	DOSETUPMENUS
TBUSYCURSOR	DOSHOWHELP
TCLIPBOARDMGR	DOTOOLBOXEVENT

```

pascal void TRApplication::DoMenuCommand(CommandNumber aCommand
{
    // =====
    // Some commands will be posted to perform actions that mus
    // The allocation cannot be allowed to fail. So we do a te
    // definition cannot be allowed to fail. This strategy is
    // command objects but don't want to leave the user twistin
    // NOTE: Don't forget to allow for this memory in your mem!
    // style in your own code.
    // =====

    Boolean oldObjectPerm;

    switch (aCommandNumber)
    {
        case cQuit:
            oldObjectPerm = AllocateObjectsFromPerm(FALSE);
    }
}
  
```

TAPPLICATION.DOMENUCOMMAND

00000000:	LINK	A6,\$FEAC
00000004:	MOVEM.L	D3/D6/D7/R3/R4,-(A7)
00000008:	MOVE.L	\$000C(A6),D7
0000000C:	MOVER.L	\$0008(A6),A4
00000010:	MOVE.L	D7,D0
00000012:	SUBQ.L	#\$1,D0
00000014:	BMI	\$00000236
00000018:	CMPL.L	#\$00000028,D0
0000001E:	BGT	\$00000236
00000022:	ADD.L	D0,D0
00000024:	MOVE.W	\$0000002C(D0.L),D0
00000028:	JMP	\$0000002A(D0.W)
0000002C:	DC.W	\$01BE
0000002E:	ANDI.B	#\$020C,A4
00000032:	ANDI.B	#\$020C,A4
00000036:	ANDI.B	#\$020C,A4
0000003A:	ANDI.B	#\$020C,A4
0000003E:	ORI.L	#\$00BC00BC,SR
00000044:	ORI.L	#\$00BC00BC,SR
0000004A:	ORI.L	#\$00BC00BC,SR
00000050:	ORI.L	#\$010C010C,SR
00000056:	MOVEP.W	\$010C(A4),D0
0000005A:	MOVEP.W	\$010C(A4),D0
0000005E:	MOVEP.W	\$010C(A4),D0
00000062:	MOVEP.W	\$010C(A4),D0
00000066:	MOVEP.W	\$010C(A4),D0

Using SourceBug

Setting breakpoints

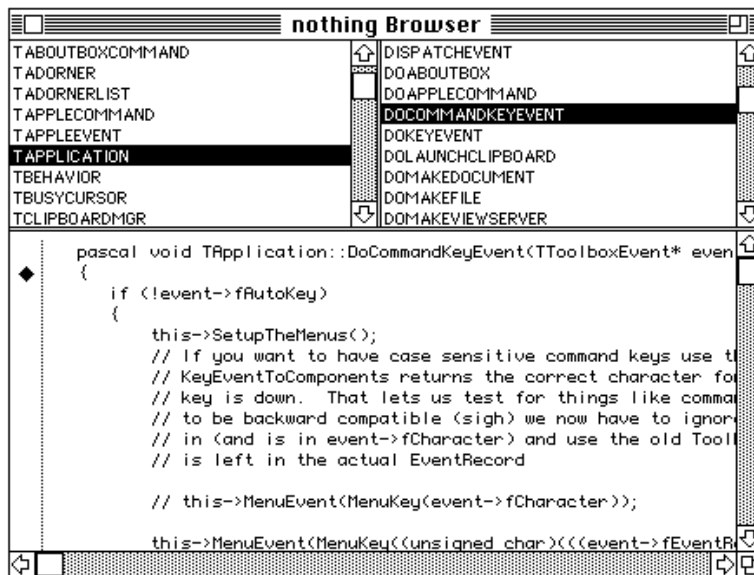
The lower pane in a Browser window (or in a Stack Crawl window—see the next section) not only displays source code but also allows you to set breakpoints to control the target application.

To set a breakpoint, move the pointer to the left

of the dotted line in the lower pane of a Browser or Stack Crawl window, where a diamond appears on the tip of the pointer. Place the pointer next to the appropriate statement and click once, putting a breakpoint on the statement.

For example, Figure 2-7 shows a breakpoint set on the first statement of the `TApplication.DoCommandEvent` method.

Figure 2-7 Breakpoint diamond icon

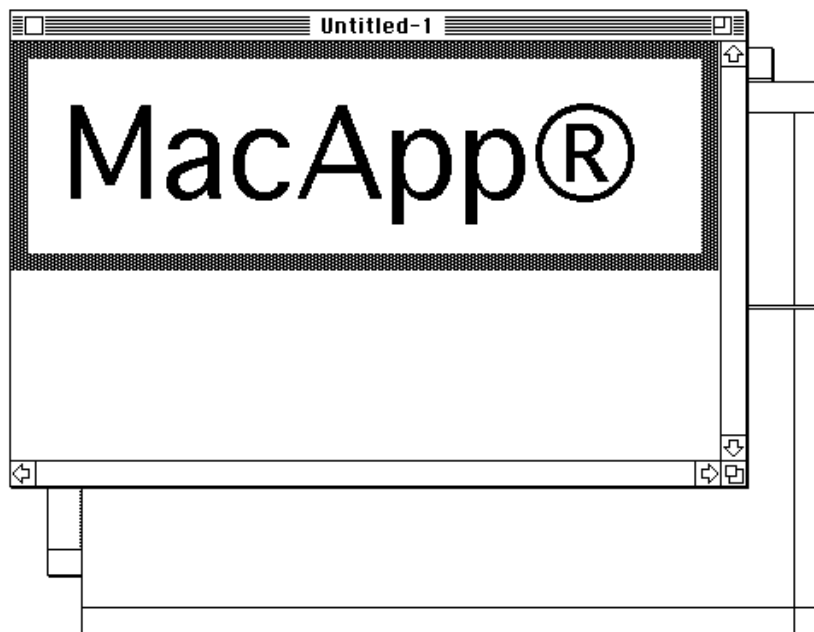


To remove a breakpoint, place the diamond pointer on top of the breakpoint marker and click once. To remove all breakpoints, choose Clear All Breakpoints from the Control menu.

Using the Stack Crawl window

The Stack Crawl window appears only when the target application is suspended. For example, assume that conditions in the Browser window are set as in Figure 2-7—that is, `TApplication` is selected in the list of classes, `DoCommandKeyEvent` is selected in the list of methods, and a breakpoint is set on the first statement in the code listing. When you launch “nothing” (the target application) by choosing Run from the Control menu (or by using the keyboard equivalent, Command-R), the “nothing” application appears in front of a blank Stack Crawl window (displaying “MacApp” with a registered trademark), as shown in Figure 2-8.

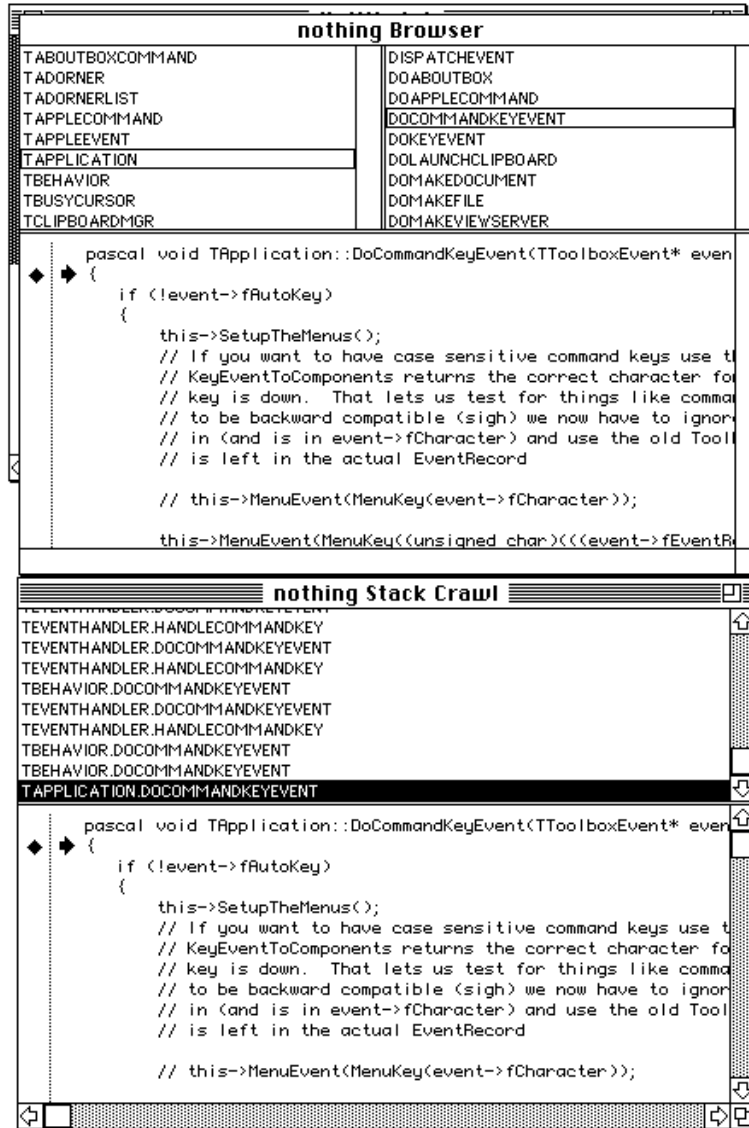
Figure 2-8 “Nothing” application display



Using SourceBug

`TApplication.DoCommandEvent`, in which the breakpoint is set, controls what happens when a user presses a Command-key combination in “nothing.” If you press a Command-key combination, such as Command-N, control is returned to SourceBug and the Stack Crawl window appears, as shown in Figure 2-9.

Figure 2-9 Stack Crawl window



Using SourceBug

NOTE

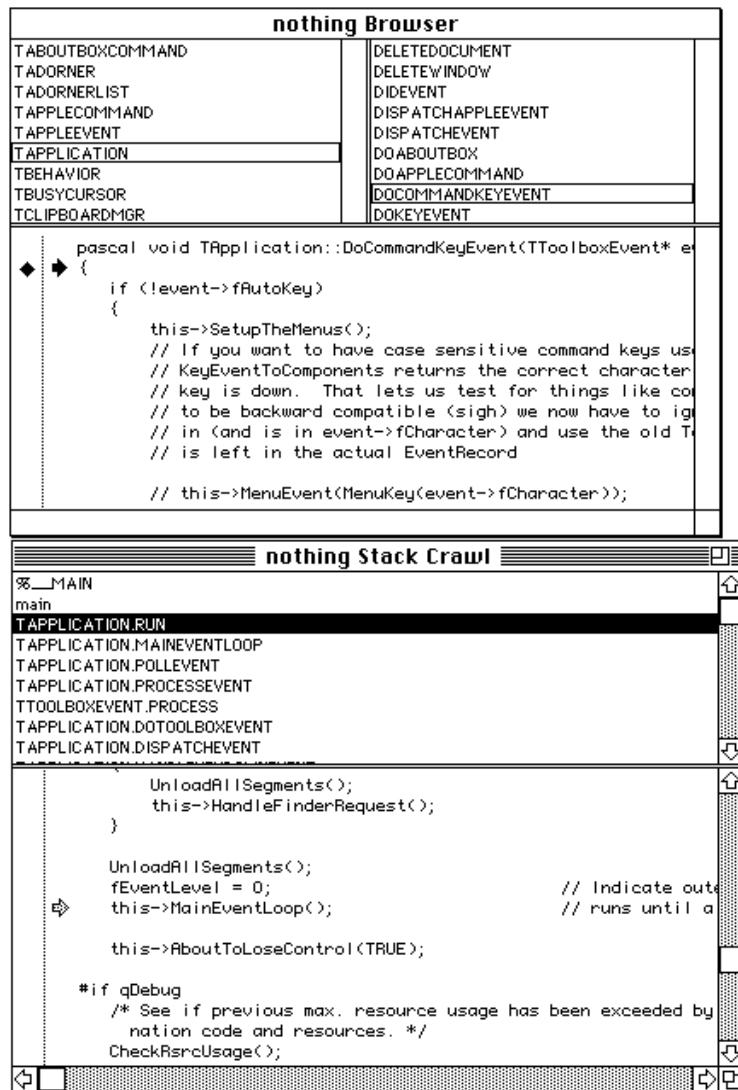
The Stack Crawl window initially covers the Browser window, but in Figure 2-9 the Stack Crawl window has been dragged down so that both windows are visible. ♦

Selecting methods (activations) on the stack

As you can see, the Stack Crawl window contains two panes: the upper one lists all the methods that are active on the stack (known as activations). The lower pane contains the source code (or disassembly) of the method highlighted in the upper pane. The right arrow identifies the program counter, or current execution point.

Note that the Stack Crawl window has no close box. It remains open unless you stop execution of the target application (with the Kill command in the Control menu) or quit the target application.

SourceBug automatically highlights the latest activation on the stack when it opens the Stack Crawl window. However, you can look at the code for any method that is active on the stack. For example, if you scroll up the list of methods and select `TApplication.Run`, the lower pane displays the `TApplication.Run` source code, as shown in Figure 2-10.

Figure 2-10 Source code for stack activation

Note that in the lower pane of the Stack Crawl window, which does not contain the program counter, the right arrow is dimmed. It points to `MainEventLoop`, which is the next activation on the stack. The dark arrow in the Browser window indicates where the program counter is located.

In some cases you can have many windows displayed, none of which shows the program counter. You can always bring the window with the program counter to the front, however, by selecting the last activation in the upper pane of the Stack Crawl window.

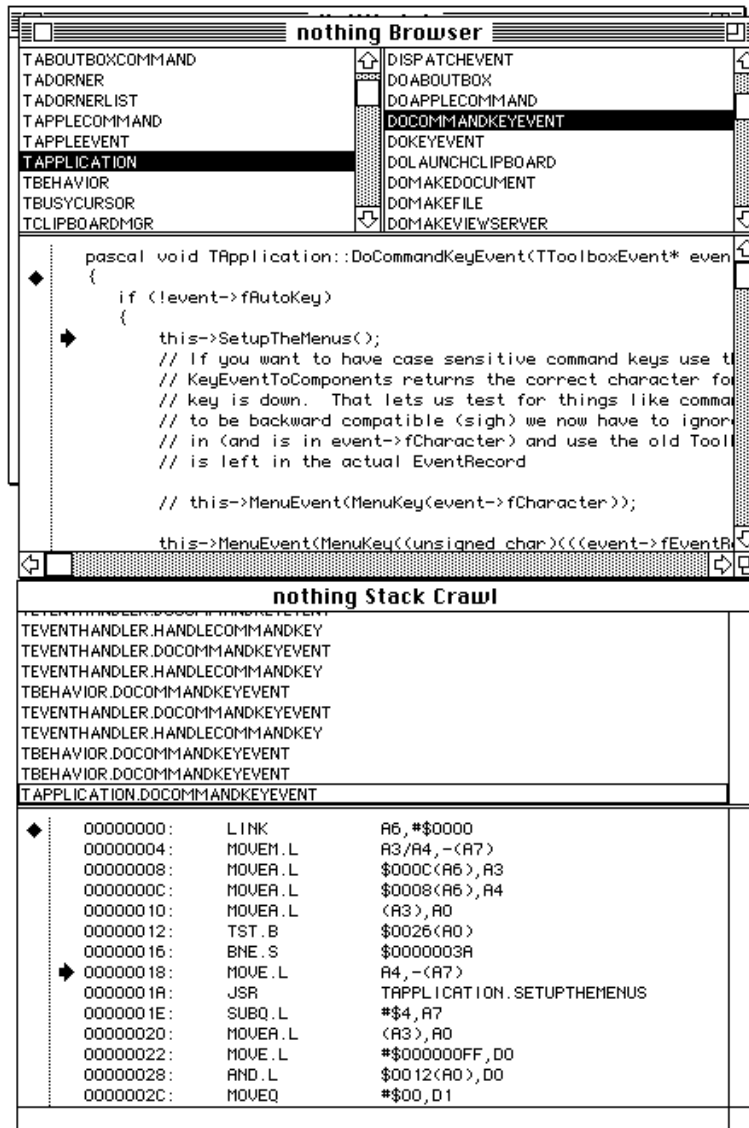
Controlling program execution

Whenever the program is suspended—for example, at a breakpoint, as in Figure 2-9—you can use the Step command in the Control menu to execute code in the target application. The way the frontmost code listing is displayed determines what Step does:

- If the frontmost code listing displays source code, Step executes one line of source code.
- If the frontmost code listing displays a disassembly, Step executes a single machine-language instruction.

To determine whether your source code is executing the machine-language instructions that you expect, you can set a breakpoint to suspend the application in a method of interest. Then create two displays of this method, one of source code and the other of a disassembly. Make the source listing the active window, and issue as many Step commands as necessary to see what machine-language instructions your source commands are executing.

For example, suppose the application is suspended at the breakpoint in `TApplication.DoCommandEvent`, as shown in Figure 2-9. If you make the Stack Crawl window display a disassembly and make the Browser window active, Figure 2-11 shows the result of stepping twice.

Figure 2-11 Source and disassembly after stepping

The Step command treats traps and subroutines as single statements—that is, it steps over them. Use the Step Into command to step into a called subroutine. Step Into does not step into A-traps.

If a procedure call is a polymorphic method call, Step Into does not step through the method-dispatching routine, but traces until dispatching is done and stops at the first line of the called method. Use the Step Out command to complete and exit the current method and return to the calling method.

Using SourceBug

Animation mode

SourceBug provides animation mode in which it continuously executes the source code by repeating a Step, Step Into, or Step Out mode. To initiate animation mode, select Animate from the Control menu. (A check mark appears by Animate in the menu.) When you issue one of the Step commands, SourceBug repeats it until you select Animate again to turn off animation mode. Whenever you issue a Step command, SourceBug updates all Variable View windows, so if you are interested in watching particular variables over a number of steps, you will find animation mode useful.

To see the effect of animation mode, turn it on and then select Step Out—be certain the program is suspended—and you can watch SourceBug move up the Stack Crawl chain.

Of course, controlling the source program is only one part of debugging. You must also look at the variables in the program while controlling its operation. The next section explains how to use variable windows in SourceBug to look at program variables.

Using variable windows

In variable windows you can display the values of variables, including simple types such as integers and Boolean variables, and complex types such as Pascal records, C structs, and classes. Use the Evaluate command (in the Inspect menu) and type in the name of a variable, or, using the Evaluate “ ” command, highlight a variable name and select Evaluate *varname*. SourceBug automatically places the highlighted name in the menu next to Evaluate.

For simple types, either Evaluate command displays the variable name and value on a single line:

```
device = 0
```

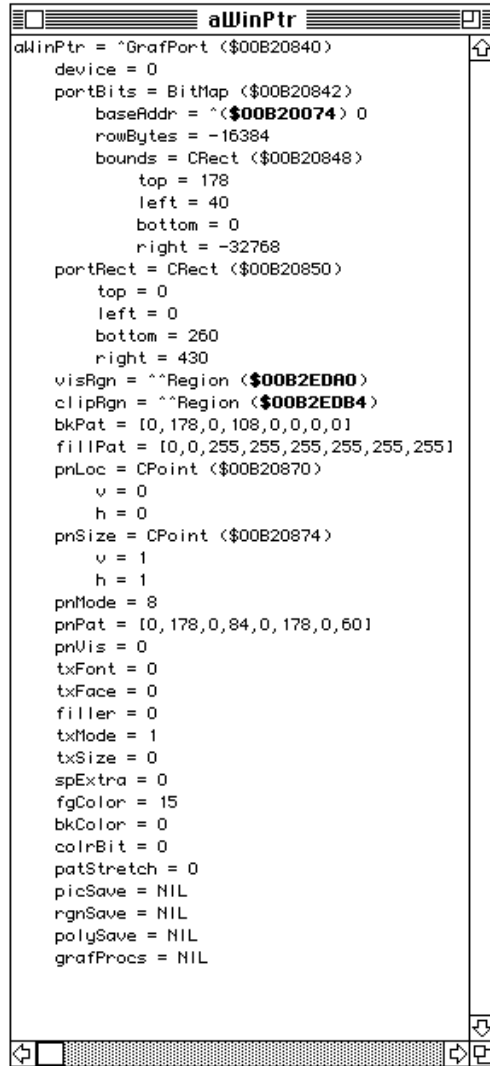
Complex types appear on several lines:

```
portRect = Rect ($0000E6B4)
    top = 0
    left = 0
    bottom = 870
    right = 640
```

Using SourceBug

If you are evaluating a variable that is a pointer or handle type, the variable window shows the object referred to by the variable. For example, `savedPort` is a pointer to a `grafPort` (in `TApplication.ExcludeWindowRegions`). If you evaluate `savedPort`, SourceBug displays the `grafPort`, as shown in Figure 2-12.

Figure 2-12 The `savedPort` variable



Using SourceBug

If a field in a record or struct is itself a pointer or handle, its address appears in boldface type in the variable window; in the example shown in Figure 2-12, `baseAddr` is a pointer and `visRgn` and `clipRgn` are handles. If you double-click one of these fields, SourceBug displays a new window showing the object referred to by the field.

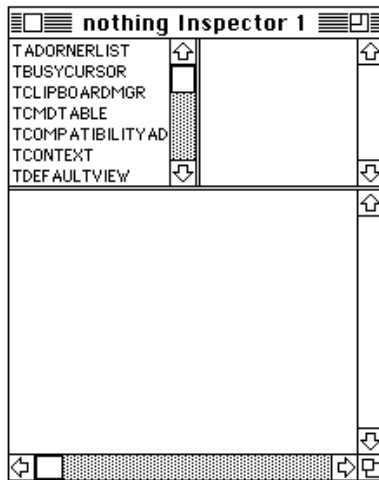
Each time SourceBug is entered (for example, after a step), it updates the values in all variable windows.

For simple types, you can use the View as commands in the Inspect menu to display a variable as a different type. (For more information about these commands, see “The Inspect Menu” in Chapter 3, “Menu Reference.”)

Using the Inspector window

The Inspector window lists all instantiated classes (classes that have objects) and allows you to display their fields. Use the New Inspector Window command in the Inspect menu to display an Inspector window. (The Inspect menu is available only when the target application is suspended.) Figure 2-13 shows an Inspector window with the target application suspended in `TApplication.DoCommandKeyEvent`.

Figure 2-13 Inspector window



The upper-left pane lists classes that have objects, and when you select a class, the upper-right pane lists its objects—actually, the hexadecimal address of the master pointer for each object.

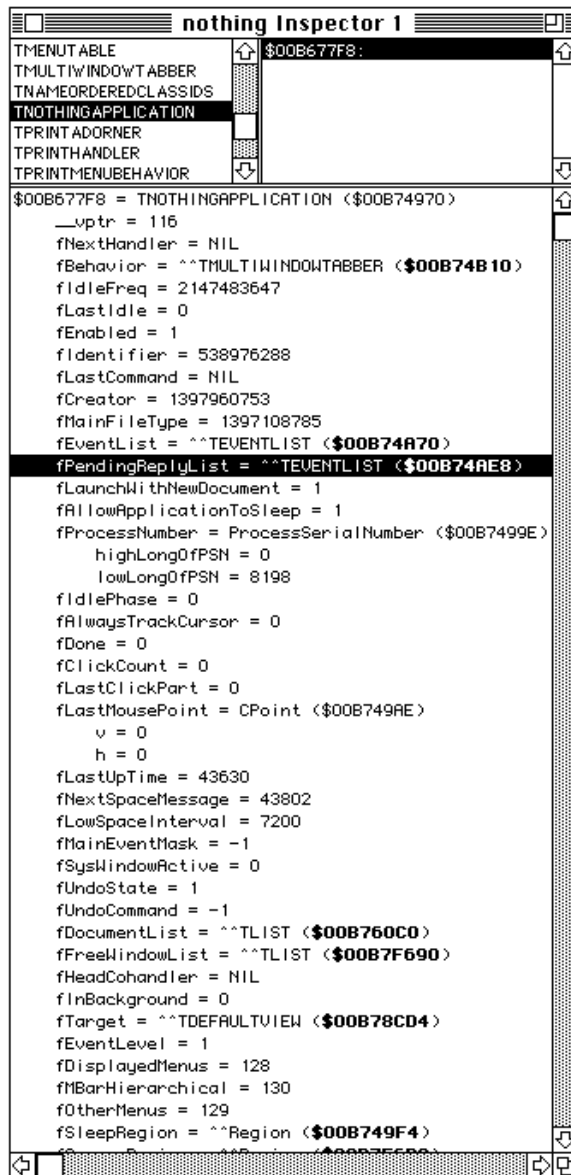
Using SourceBug

Displaying the fields of an object

When you select an object in the upper-right pane, SourceBug displays the fields of that object in the lower pane.

For example, if you create a new Inspector window and select the only instance of `TNothingApplication`, the list of fields shown in Figure 2-14 appears.

Figure 2-14 Inspector variable window



Using SourceBug

As in the variable window described earlier, you can double-click any item in boldface type to bring up another variable window with more information.

Using SourceBug

Menu Reference

Menu Reference

This chapter describes the SourceBug menus. It describes in detail all the menus and menu commands that are unique to SourceBug and provides an overview of menu commands, such as Cut, Copy, and Paste, that are standard to most Macintosh applications.

The File menu

The commands in the File menu, shown in Figure 3-1, allow you to select an application to debug, close windows, save the contents of the Log window, and quit SourceBug.

Figure 3-1 File menu

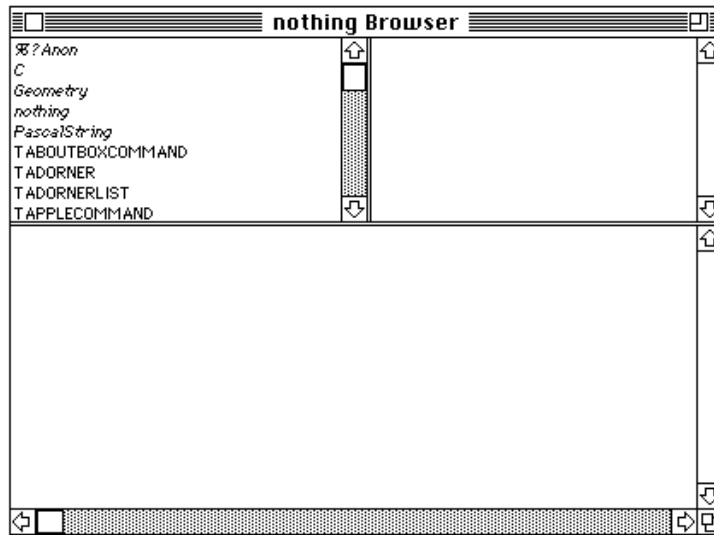


File	
Open...	⌘O
Close	⌘W
Save Log As...	
Quit	⌘Q

Open... (Command-O)

The Open command displays a Standard File dialog box from which you can select an application to debug (the target application). When you select an application, SourceBug opens up a Browser window with the name of the application, as shown in Figure 3-2.

Menu Reference

Figure 3-2 Browser window for the “nothing” application

If the dates on the source files and the symbol file don’t match, SourceBug displays an error message; if it cannot find the source files, SourceBug opens a dialog box asking you to locate them.

You can open more than one application to debug, if you wish, as long as enough memory is available.

Close (Command-W)

The Close command closes the active window.

▲ **WARNING**

If you close the Browser window for a particular application, SourceBug removes that application as the target and closes all windows containing information about that application. ▲

Save Log As...

The Save Log As command displays a Standard File dialog box allowing you to save the contents of the Log window to an MPW text file. This command is dimmed if the Log window is empty; however, you can save the contents of the Log window whether it is visible or hidden.

Quit (Command-Q)

The Quit command quits SourceBug and any target applications.

The Edit menu

The Edit menu, shown in Figure 3-3, provides standard Macintosh editing functions, such as undoing, cutting, copying, pasting, clearing, and selecting everything in a window; and showing the contents of the Clipboard. It also allows you to see the contents of the registers and of the floating-point unit (FPU) registers, and to show the Log window. This section describes the Edit menu commands that are unique to SourceBug (or that have a function unique to SourceBug) under separate headings; the editing commands that are not unique to SourceBug are described under the heading “Standard Macintosh Editing Commands.”

Figure 3-3 Edit menu

Edit	
Undo	⌘Z
Cut	⌘H
Copy	⌘C
Paste	⌘V
Clear	
Select All	⌘A
Show Clipboard	
Show Registers	
Show FPU Registers	
Show Log Window	

Undo/Redo (Command-Z)

The Undo (or Redo) command removes or resets a breakpoint, depending on the action you last took. That is, if you previously set a breakpoint, Undo removes it (and Redo sets it again); if you removed a breakpoint, Undo sets it again (and Redo removes it). Note that Undo applies to breakpoints that you set manually and to those set with the Set Breakpoint at Failure command in the Control menu.

After you select Undo, the command changes to Redo.

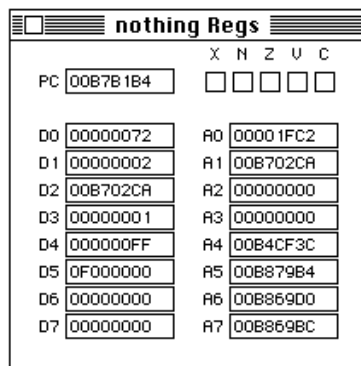
Standard Macintosh editing commands

The Edit menu includes the Cut, Copy, Paste, Clear, Select All, and Show Clipboard commands. Because SourceBug does not have a text editor, the Cut, Paste, and Clear commands are always dimmed. In a code listing, you can use Select All to select all the text (or use the mouse to select part of the text), use the Copy command to copy the selection, and then use the Paste command to paste it into a different application. In a variable window, Copy always copies the entire window, even if you have selected a single line. (For more information on variable windows, see “Using Variable Windows” in Chapter 2, “Using SourceBug.”)

Show Registers

The Show Registers command opens a window that displays the contents of the program counter, the data registers (D0–D7), the address registers (A0–A7), and the status of the condition code register (bits X, N, Z, V, and C), as shown in Figure 3-4. Whenever SourceBug is reentered (for example, when you issue a Step command), SourceBug updates the display in this window.

Figure 3-4 Registers for the “nothing” application



Menu Reference

To change the value in a register, highlight the appropriate register using the mouse or the Tab key, and type in a new value (SourceBug adds leading zeroes as necessary), or, after highlighting the register, move left and right with the arrow keys and use the Delete key to delete one or more numbers if you wish. Press Return or Enter after typing a new value (or press Tab to move to the next register). You can change as many registers at one time as you wish.

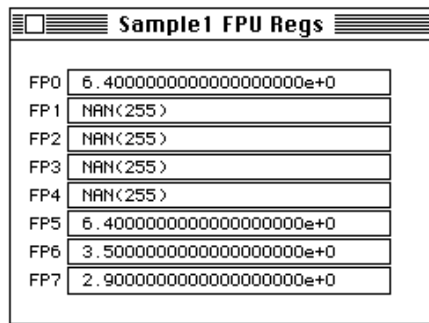
You can also set or clear bits in the condition code register by selecting or deselecting the appropriate box (X, N, Z, V, or C).

If you change the value of the program counter, the A6 register, or the A7 register, SourceBug updates the Stack Crawl window.

Show FPU Registers

The Show FPU Registers command opens a window that displays the contents of the floating-point unit (FPU) registers, as shown in Figure 3-5. Whenever SourceBug is reentered (for example, when you issue the Step command), SourceBug updates the display in this window.

Figure 3-5 Sample FPU registers



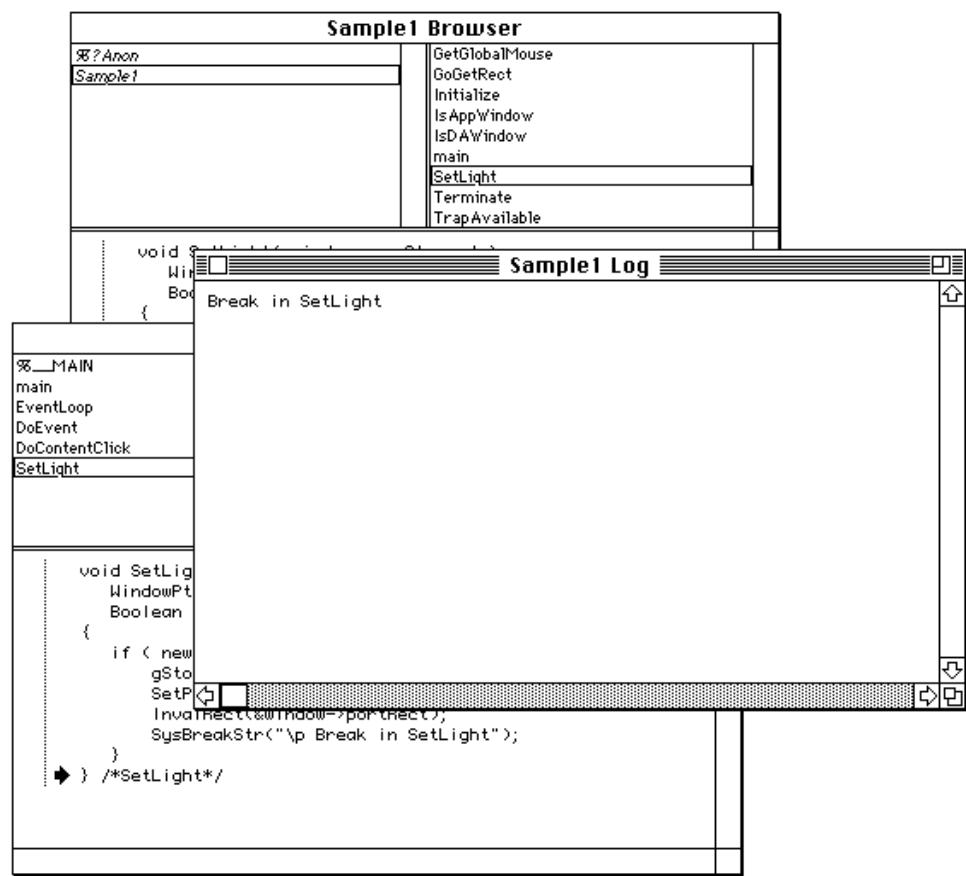
Register	Value
FP0	5.4000000000000000e+0
FP1	NAN(255)
FP2	NAN(255)
FP3	NAN(255)
FP4	NAN(255)
FP5	5.4000000000000000e+0
FP6	3.5000000000000000e+0
FP7	2.9000000000000000e+0

Show Log Window

The Show Log Window command opens a Log window (as shown in Figure 3-6) that displays any debugging statements in the application source code that are encountered as you run the application from SourceBug. Whenever SourceBug is reentered (for example, when you issue the Step command), SourceBug updates the display in this window. See the Appendix, “Using Manual Breakpoints With SourceBug,” for information on putting different kinds of debugging statements into your source code.

You can highlight text in the Log window and copy it to the Clipboard (to paste to another application), but you cannot edit the text in any way. You can also use the Save Log As command in the File menu to save the contents of the Log window to an MPW text file (for example, if you want to print the messages that appear in the Log window).

Figure 3-6 Sample Log window



The Class menu

The commands in the Class menu, shown in Figure 3-7, allow you to choose how to display classes in the Browser window: in alphabetical order or according to class hierarchy.

Figure 3-7 Class menu



Alphabetical Classes

The Alphabetical Classes command provides an alphabetical display of the classes in the Browser window's upper-left pane, which allows you to quickly locate classes in the list.

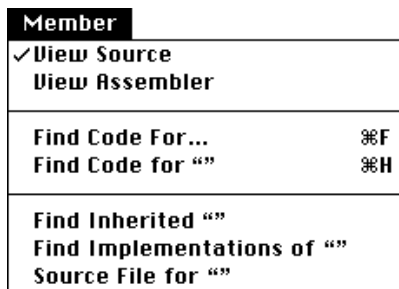
Hierarchical Classes

The Hierarchical Classes command provides a hierarchical display of the classes shown in the Browser window's upper-left pane, which allows you to view the relationships between the classes in the target application.

The Member menu

The commands in the Member menu, shown in Figure 3-8, allow you to determine how you want to view source code, detect inheritance in methods, and locate source code.

Figure 3-8 Member menu



View Source

The View Source command displays the code that appears in the active SourceBug window as source code.

View Assembler

The View Assembler command displays a disassembly of the code that appears in the active SourceBug window.

Find Code For... (Command-F)

The Find Code For command opens a dialog box in which you type the name of a routine or method; SourceBug finds the source code for this method and displays it in a new window.

Menu Reference

If SourceBug cannot find the routine, it returns the message “Not Found.” If SourceBug finds the routine but cannot locate its source code, it provides a Standard File dialog box asking you to locate the source. If you cannot locate the source for SourceBug (or if there is no symbolic information for the routine), SourceBug displays a disassembly of the routine.

Find Code for “ ” (Command-H)

The Find Code for “ ” command finds the source code for a routine whose name you have highlighted in the active window’s code listing. When you highlight a routine name, SourceBug replaces the quotation marks in the command name with the name of the routine. SourceBug displays the code in a new window.

Find Inherited “ ”

The Find Inherited “ ” command finds and displays the superclass implementation of the method displayed in the window’s code listing. (SourceBug replaces the quotation marks with the name of the method.) For example, `TView` is a subclass of `TEvtHandler`, and both of these classes have a `DoMenuCommand` method. When the source code for `TView.DoMenuCommand` is displayed and you choose Find Inherited “ ”, SourceBug displays the source code for `TEvtHandler.DoMenuCommand`. If the highlighted method has no implementation in its superclass, this command is dimmed.

Find Implementations of “ ”

The Find Implementations of “ ” command provides a hierarchical list of all classes that implement the method that is displayed in the frontmost window’s code listing. (SourceBug replaces the quotation marks with the name of the method.) If the method in the frontmost code listing doesn’t belong to any classes, this command is dimmed.

When you select a class from the list provided by this command, SourceBug displays the source code for the specified method in that class.

Source File for “ ”

The Source File for “ ” command opens a dialog box listing the name of the source file that contains the method displayed in the frontmost window’s code listing. (SourceBug replaces the quotation marks with the name of the method.)

The Control menu

The commands in the Control menu, shown in Figure 3-9, allow you to control the target application by setting and clearing breakpoints, executing the source code, and stopping execution of the program. You can also turn control over to a low-level debugger, such as MacsBug.

Figure 3-9 Control menu

Control	
Run	⌘R
Kill	⌘K
Switch to Low-level Debugger	
Step	⌘S
Step Into	⌘I
Step Out	⌘P
Set Breakpoint at Failure	
Clear All Breakpoints	

Run (Command-R)

The Run command launches the target application if it is not already running; Run resumes execution of a suspended target application at the current program counter.

Kill (Command-K)

The Kill command stops execution of the target application. Note that Kill does not remove the current application as the target and that the Browser window remains open. However, Kill does close the Stack Crawl window and any Inspector windows that are open.

Switch to Low-level Debugger

The Switch to Low-level Debugger command turns control of the target application over to your low-level debugger. This command is enabled if the target application is suspended and the _Debugger A-trap is implemented, indicating the existence of a low-level debugger, such as MacsBug. The Process Manager performs a full context switch to the target application and then mimics the pressing of the programmer's interrupt switch.

When you enter the low-level debugger, the program counter is positioned at the next instruction in the target application. You can return control to SourceBug by resuming execution of the target application; for example, in MacsBug, issue the G (Go) command.

Step (Command-S)

The Step command executes the target application one source statement or one machine-language instruction at a time. The display in the active window determines what Step does:

- If the active window displays source code, Step executes one line of source code.
- If the active window displays a disassembly, Step executes a single machine-language instruction.

The Step command treats traps and subroutines as single statements—that is, it steps over them.

Step Into (Command-I)

The Step Into command operates the same as the Step command, except that it steps into procedure calls and stops at the first line of the called routine.

Step Into does not step into A-traps. If you attempt to step into a polymorphic method call, SourceBug does not step through the method-dispatching routine but instead traces until method dispatching is done; SourceBug then stops at the first line of the called method.

Step Out (Command-P)

The Step Out command resumes execution of the current application, completes and exits the current routine, and returns to the calling routine.

Animate

Enables animation mode, in which SourceBug continuously executes the next Step, Step Into, or Step Out command that you choose. A check mark appears next to the Animate command when you select it to enable animation mode. The mouse is still active in animation mode, so you can select Animate again to disable animation mode and stop execution of the command that is currently executing.

Set Breakpoint at Failure

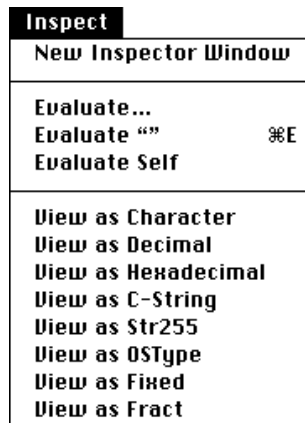
The Set Breakpoint at Failure command sets a breakpoint that suspends program operation rather than calling the MacApp failure routine.

Clear All Breakpoints

The Clear All Breakpoints command removes all breakpoints, including breakpoints you have set manually and with the Set Breakpoint at Failure command.

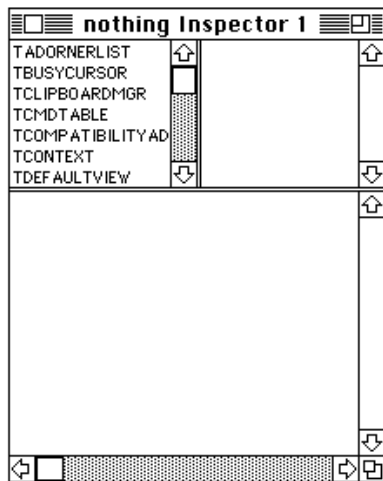
The Inspect menu

Using the commands in the Inspect menu, shown in Figure 3-10, you can open an Inspector window, which allows you to look at the fields of instantiated classes, and display the contents of variables. This entire menu is dimmed until you run and suspend the target application.

Figure 3-10 Inspect menu

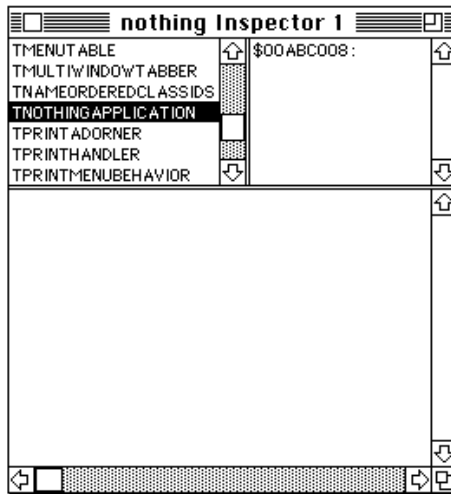
New Inspector Window

The New Inspector Window command opens an inspector window, which contains three panes. In the upper-left pane, as shown in Figure 3-11, SourceBug lists all the classes that have been instantiated at least once.

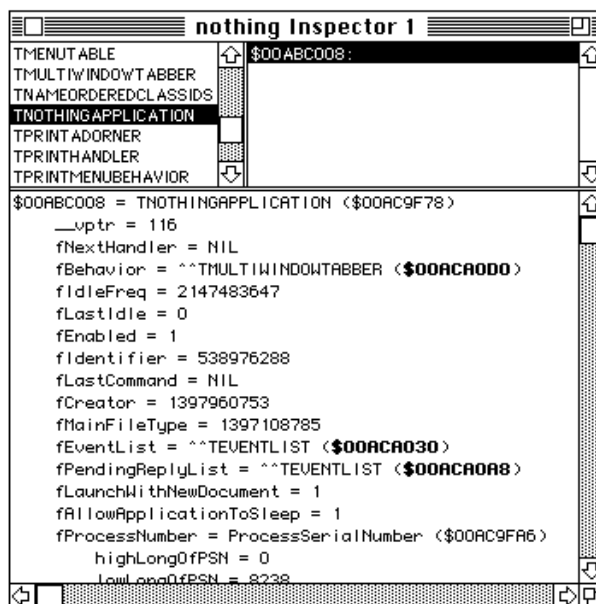
Figure 3-11 A new Inspector window

When you select one of these classes, SourceBug displays the hexadecimal address of the master pointer for each object of that class (if there are any) in the upper-right pane, as shown in Figure 3-12.

Menu Reference

Figure 3-12 An Inspector window with a class selected

When you select one of the objects in the upper-right pane, SourceBug displays the fields of that object and their contents, as shown in Figure 3-13.

Figure 3-13 Inspector window showing fields

If the address for a field is displayed in boldface, for example, the address of `fEventList = ^^TEventList` in Figure 3-13, you can double-click it to see its contents. SourceBug displays a new window that lists the selected field's name and its contents.

Evaluate...

The Evaluate command opens a dialog box in which you type the name of a variable, field, or structure. SourceBug displays a window that contains the value of the variable, field, or structure you specify.

Evaluate “ ” (Command-E)

The Evaluate “ ” command displays the value of the variable, field, or structure that you have highlighted in a window. SourceBug displays the variable’s name and value in a new window.

Evaluate Self

When you choose Evaluate Self with the Browser window active, the Evaluate Self command searches within the current scope of the frontmost code listing for a local variable named `SELF` (Pascal) or `this` (C++) and uses it to display the fields or data members of the object associated with the currently executing method. You can also highlight a particular method in the Stack Crawl window; then, when you choose Evaluate Self, it displays information about the fields or data members of the object associated with the highlighted method.

View as

The various versions of this command allow you to view a simple type as a character, decimal, hexadecimal type, and so forth.

The Windows menu

The windows menu lists all of the windows that are currently displayed in SourceBug. Choosing a name in the list makes that window the active window and brings it to the front.

Menu Reference

Using Manual Breakpoints With SourceBug

This appendix describes how to put break and print statements into your application's source code manually for use with SourceBug.

Putting breakpoints into your source code manually

SourceBug allows you to put breakpoints into your application source code manually using certain MPW interface routines or specific routines defined in the files in the MacApp CPlusIncludes directory. You can attach a message to the breakpoint by passing a text string as an argument to the break statement that executes it, or you can specify a message to print without setting a breakpoint by putting in a separate print statement.

If SourceBug is running when a break statement is executed in your application, SourceBug is entered, and the application breaks in the routine containing the statement. If a text string is passed to the break statement or if a print statement is executed in your application, SourceBug displays the text in the Log window.

You can select an application containing break or print statements as the target and then launch it from within SourceBug (using the Run command). When a break or print statement is encountered, SourceBug is reentered (in the case of a break statement), and the message, if there is one, is displayed in the SourceBug Log window.

Another way to break into SourceBug from your application is to begin with SourceBug running and launch your application from the Finder (or from MPW). When a break or print statement is encountered, SourceBug is entered, and the message, if any, is displayed in the Log window. If you launch the application in this manner and SourceBug is not running, the application breaks into MacsBug (or into another debugger, such as SADE, if it is running).

The next section describes the specific statements to use to insert breakpoints manually into source code.

Break and print statements using SysBreak

The MPW interface files, `Types.h` (C interfaces) and `Types.p` (Pascal interfaces), and the MacApp CPlusIncludes file, `Types.h`, define three routines, `SysBreak`, `SysBreakStr`, and `SysBreakFunc`, that enable you to put break statements and messages into your source code, as follows:

<code>SysBreak</code>	Breaks without a message—for example: <code>SysBreak ();</code>
<code>SysBreakStr</code>	Breaks and allows you to specify a message—for example: <code>SysBreakStr('Break in DoMenuCommand');</code>
<code>SysBreakFunc</code>	Allows you to specify a message but does not break—for example: <code>SysBreakFunc('Break in SetLight'); SysBreakFunc</code>

Because the message parameter for `SysBreak` and `SysBreakFunc` is defined as a Pascal string in the MPW interface files, when using either of these routines in a C program, you must begin the string with `\p`—for example:

```
SysBreakStr( "\p Break in DoMenuCommand" );
```

However, MacApp 3.0 and later provides support for converting between C strings and Pascal strings, so you can enter it as a C string if you wish:

```
SysBreakStr( "Break in DoMenuCommand" );
```

Print statements using printf

You can use the three `SysBreak` routines in any applications written under MPW, including applications written with MacApp. In addition, because the mechanism in MacApp to support `SysBreak` uses low-level

I/O redirection, in MacApp applications you can use statements such as `printf`, `fprintf`, `sprintf`, and the like that make use of standard out or standard error (`stderr`) to put debugging messages into your source code. The various `printf` statements work exactly like `SysBreakFunc`:

if SourceBug is running and your application executes a `printf` statement, SourceBug displays the message in its Log window.

The advantage of `printf` over `SysBreakFunc` is that with `printf` you can format the messages that are sent to the SourceBug Log window (although you cannot break into the application with `printf`). For example,

```
fprintf(stderr, "%s the value of gStopped is\n", gStopped);
```

puts a line break after each statement. If your application executes this statement or other print statements numerous times, each message appears on a separate line in the Log window.

NOTE

MacApp uses a similar mechanism when compiling with debugging on (`-debug on` flag) to create `fprintf` statements that send error messages at compile time if your application fails basic error checking. ♦

Index

A

activations on stack 19–20
address registers 33
Alphabetical Classes command 12, 36
A-traps 22
 and stepping 39

B

breakpoints
 inserting manually 45
 removing 16, 39
 setting 16
break routines 46
Browser window 10
 contents of 10–12
 effect of closing 10, 31

C

classes
 listed in Browser window
 11, 36
 listed in Inspector window
 26, 40
Class menu 35–37
Clear All Breakpoints command 16,
 39
Clear command 33
cloning code panes 14
Close command 31
code listings 12
 cloning 14
 disassembly 12, 36
 highlighting code in 13
code panes, cloning 14

commands
 Alphabetical Classes 12, 36
 Clear 33
 Clear All Breakpoints 16, 39
 Close 31
 Copy 13, 33
 Cut 33
 Evaluate 42
 Evaluate “ ” 42
 Evaluate Self 43
 Find Code For 37
 Find Code for “ ” 37
 Find Implementations of
 “ ” 12–13, 37
 Find Inherited “ ” 37
 Hierarchical Classes 12, 36
 Kill 19, 38
 New Inspector Window 25, 40
 Open 31
 Paste 33
 Quit 32
 Redo 33
 Run 17, 38
 Save Log As 32
 Select All 33
 Set Breakpoint at Failure 39
 Show Clipboard 33
 Show FPU Registers 34
 Show Log Window 34
 Show Registers 33
 Source File for “ ” 37
 Step 21, 39
 Step Into 22, 39
 Step Out 22, 39
 Switch to Low-level Debugger 38
 Undo 33
 View as 43
 View Assembler 12, 36
 View Source 12, 36
compilation units 11
condition code registers 33
Control menu 38–39
Copy command 13, 33

current execution point, finding 20
Cut command 33

D

data registers 33
diamond pointer 16
disassembly, displaying 12
dummy classes 11

E

Edit menu 32–34
Evaluate command 42
Evaluate “ ” command 42
Evaluate Self command 43
expanding panes 10

F

fields, displaying 26, 42
File menu 30–32
Find Code For command 37
Find Code for “ ” command
 14, 37
Find Implementations of “ ”
 command 12–13, 37
Find Inherited “ ” command 37
floating-point unit (FPU) registers 34
fprintf 46, 47

H

handles, evaluating 24–27
hardware requirements 2
Hierarchical Classes command 12, 36

I

inherited methods, finding 12, 37
 Inspect menu 40–43
 Inspector window 25
 installing SourceBug 3
 instantiated classes, listing 25
 interface files 46
 italics, in Browser window 11

K

Kill command 19, 38

L

launching SourceBug 9
 Log window 34, 45
 low-level debuggers 4, 38

M

MacApp classes 11
 MacApp source files 11
 machine-language instructions 39
 correlating to source statements 21
 Member menu 36
 memory requirements 2, 4
 menu commands. *See* commands
 menus
 Class 35–37
 Control 38–39
 Edit 32–34
 File 30–32
 Inspect 40–43
 Member 36
 Windows 43
 messages
 displaying in SourceBug 45
 placing in source code 45
 method dispatching
 and stepping 22, 39
 method-dispatching routines 22
 methods
 implementations of 37
 inherited 37
 listed in Browser window 11
 polymorphic 22, 39

source code displayed in Browser
 window 12

MPW tools 5
 MultiFinder versions 3

N

New Inspector Window
 command 25, 40
 “nothing” sample application 8, 9

O

Open command 31

P

panes, expanding 10
 Paste command 33
 %?Anon class 11
 pointers, evaluating 24–27
 polymorphic method calls 22
 and stepping 39
 printf 46, 47
 print routines 46, 47
 program counter 19, 33
 finding 20
 icon for 19

Q

Quit command 32

R

Redo command 33
 registers, displayed in window 33–34
 removing breakpoints 16, 39
 removing target applications 10
 requirements, hardware and software 2
 right arrow icon 19
 routines, in Browser window 11
 Run command 17, 38

S

SADE 5
 Save Log As command 32
 scripts 5
 Select All command 33
 selecting text 13
 SELF variable 43
 Set Breakpoint at Failure command 39
 setting breakpoints 16
 Show Clipboard command 33
 Show FPU Registers command 34
 Show Log Window command 34
 Show Registers command 33
 software requirements 2
 source code, displaying 12, 37
 Source File for “ ” command 37
 source statements, correlating to
 machine-language instructions 21
 stack, activations on 19–20
 Stack Crawl window 17–23
 Standard File dialog box 9
 Step command 21, 39
 Step Into command 22, 39
 Step Out command 22, 39
 structures, displaying values of 42
 subroutines
 stepping into 22
 stepping over 22
 Switch to Low-level Debugger
 command 38
 SysBreak 46
 SysBreakFunc 46
 SysBreakStr 46

T

target applications 9
 removing 10
 text, selecting 13
 this variable 43
 Types.h file 46

U

Undo command 33

V

variables, displaying values of 23–27
 watching value of 25
View as commands 43
View Assembler command 12, 36
View Source command 12, 36

W

windows, Inspector 25
Windows menu 43

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Proof pages were created on an Apple LaserWriter IINTX printer. Final pages were created on the Varityper VT600 imagesetter. Line art was created using Adobe Illustrator. PostScript®, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type is New Aster and display type is Frutiger. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.

WRITER

Michael Kline

DEVELOPMENTAL EDITOR

Anne Szabla

ILLUSTRATOR

Sandee Karr

PRODUCTION SUPERVISOR

Teresa Lujan