



N e w t o n C + + T o o l s

Newtsbug User's Guide



1.01

February 21, 1997

© Apple Computer, Inc. 1997



Apple Computer, Inc.

© 1997 Apple Computer, Inc.

All rights reserved.

No part of this publication or the software described in it 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., except in the normal use of the software or to make a backup copy of the software and any documentation provided on CD-ROM. Printed in the United States of America.

The Apple logo is a registered trademark of Apple Computer, Inc. Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for licensed Newton platforms.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for printing or clerical errors.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, AppleTalk, eMate, Espy, LaserWriter, the light bulb logo, Macintosh, MessagePad,

Newton, Newton Connection Kit, and New York are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Geneva, NewtonScript, Newton Toolkit, and QuickDraw are trademarks of Apple Computer, Inc. Acrobat, Adobe Illustrator, and PostScript are trademarks of Adobe Systems Incorporated, which may be registered in certain jurisdictions.

CompuServe is a registered service mark of CompuServe, Inc.

FrameMaker is a registered trademark of Frame Technology Corporation.

Helvetica and Palatino are registered trademarks of Linotype-Hell AG and / or its subsidiaries.

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

Microsoft is a registered trademark of Microsoft Corporation.

Windows is a trademark of Microsoft Corporation.

QuickView™ is licensed from Altura Software, Inc.

Simultaneously published in the United States and Canada.

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manual or in the media on which a software product is distributed, ADC will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to ADC.

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

Chapter 1	Introduction	1-1
	System Requirements	1-1
	Features	1-2
Chapter 2	Getting Started	2-1
	What You Need	2-1
	Beginning Debugging	2-3
	Making Sure Your Package Symbols are Available	2-4
	Problem Alert Dialogs	2-5
	Ending a Debugging Session	2-5
	Files Newtsbug Needs to Find	2-5
	Naming the Package File So Newtsbug Can Find It	2-5
	Handling the .sym File	2-6
	If You Aren't Getting Symbols For Your Package	2-6
Chapter 3	Newtsbug Menus	3-1
	File Menu	3-1
	Open (x-O)	3-2
	Close (x-W)	3-2
	Save As...	3-2
	Load Memory...	3-2
	Save Memory...	3-2
	Frozen Newton Reconnect...	3-3
	Quit (x-Q)	3-4
	Edit Menu	3-4
	Copy Target Screen	3-4
	Register Names...	3-4
	Preferences...	3-5
	Stop on Debug traps	3-6
	Beep on Stops	3-6
	Add Time Stamps	3-6
	Log Breaks	3-6
	Log Stdio	3-6
	Log Listener	3-7
	Async Prints	3-7
	Font	3-7

Number of 4K Pages for Breakpoints	3-7
Baud Rate	3-8
Modem Port or Printer Port	3-8
Commands Menu	3-8
Go (x-G)	3-9
Stop (x-.)	3-9
Rerun (x-R)	3-9
Clear All Breakpoints	3-10
Clear All Breakpoint Hits	3-10
Disable/Enable Breakpoints	3-10
Step Into (x-S), Step Over (x-T)	3-10
Convert... (x-N)	3-10
DownLoad Package...	3-11
Command Keys for the Listener	3-11
Procedures Menu	3-12
Procedure... (x-P)	3-12
Define...	3-13
Memory Menu	3-14
Memory...	3-14
Frame...	3-15
Find...	3-18
Heap	3-18
Script Heap	3-19
Smash Heap Tags	3-19
Sort by Address	3-19
Sort by Size	3-19
Sort by Task	3-20
Windows Menu	3-20
Status	3-20
CPU	3-20
The FPE Registers	3-20
Stack Trace	3-20
Stack Trace From...	3-21
stdin/stdout/stderr	3-21
NewtonScript Listener	3-21
Config Menu	3-21
Stop on Aborts	3-21
Default Stdio On	3-21
Enable Stdout	3-21
Enable Package Symbols	3-22

Chapter 4	Newtsbug Windows	4-1
	Status Window	4-1
	Go and Step Buttons	4-2
	Rerunning	4-2

	Handy Hex Converter	4-3
	CPU Window	4-3
	FPE Registers Window	4-5
	Stack Trace Window	4-5
	Stdin/Stdout/Stderr Window	4-6
	NewtonScript Listener Window	4-6
	Code Windows	4-6
	Opening Code Windows	4-7
	Code Window Contents	4-8
	Procedure Browser	4-8
Chapter 5	Using Newtsbug	5-1
	Breakpoints	5-1
	The Breakpoint Conditions Dialog	5-2
	Expressions	5-4
	Shortcuts	5-4
	General Shortcuts	5-4
	In the CPU Window	5-5
	In Code Windows	5-5
	In the NewtonScript Listener Window	5-5
	Customizing NewtonScript Listener Shortcuts	5-5
	In the Stack Trace Window	5-6
	Debugging Examples and Tips	5-6
	Converting Values	5-6
	Examining Parameters	5-6
	Stack Frames	5-7
	Compiler Idioms	5-7
	How to find out the Real PC	5-8
	Debugging Layout and Placement	5-9

Introduction

Newtsbug is the low-level debugger for Newton development. You need to use it if you want to debug C, C++, or ARM Assembler code that runs on a Newton device.

Newtsbug runs on Mac OS-based computers.

Since Newtsbug is a low-level debugger, it works on compiled code, and does not display source code. It does, however, display higher-level language and Assembler symbols. Familiarity with ARM Assembler is very helpful in using Newtsbug. You may want, at least, to obtain the Assembler documentation.

You use Newtsbug by connecting a Newton to the Mac OS-based computer using a serial cable.

If you are developing code using the Newton C++ Tools and NTK that you want to debug using Newtsbug, you may find there are many times in your development cycle when you want to run both NTK and Newtsbug, and you need to download new versions of your program. The best way to do that is to download your program using Newtsbug rather than NTK. Newtsbug's Commands menu has a Download Package command, with the shortcut ⌘-2, which is equivalent to the NTK Inspector's command that downloads a package.

System Requirements

Newtsbug has the following system requirements:

- Any PowerPC computer running the Mac OS will work well. For 680x0 Mac OS computers, we recommend that you use a model that has a 68040, although you can use a slower model.
- The operating system must be Mac OS version 7.0 or later.
- You must be running in 32-bit mode.
- You need at least 16 megabytes of memory.

- You need a monitor screen that can display 640-by-480 resolution or better. It can be black and white, gray scale, or color.

Features

This section lists all the features supported by Newtsbug.

- Execution control
 - Stop the Newton; when the Newton is stopped, everything except possibly some low-level hardware code is stopped
 - Set breakpoints in packages or in ROM
 - Set breakpoints with conditional expressions (e.g. `R0== 0x00000000`)
 - Step and step over in ROM or packages
- Exceptions Handling
 - Stop on aborts
- Memory Related
 - See heaps
 - See memory
 - Modify memory
- Code Related
 - See code (ROM and packages)
 - Change instructions
- Stdin/stdout/stderr
 - See messages in stdin/stdout/stderr window
 - Log files
- NewtonScript Listener window
 - Cut, Copy, Paste, Clear, Select All, Undo
 - Quick keys (⌘-3 through ⌘-9), which can be customized
 - All NewtonScript `print` calls are output to this window
 - Time stamp for each line of text from the Newton
 - Window contents are automatically saved in a file when session is closed
 - Log files
- Choose any available serial channel
 - On the Newton, use the built-in serial port, or use a serial port on a PCMCIA card
 - On the Mac OS computer, use either the modem or printer port
- Error checking
 - CRC checking for packets sent between Newton and Newtsbug
 - Retransmit on error
- Load packages

Introduction

- Start debugging after the Newton crashes
 - See and modify registers
 - See FPE registers
 - Stack trace
 - Copy the Newton screen
 - CPU window
- The following features can only be used with a flash-ROM Newton.

Introduction

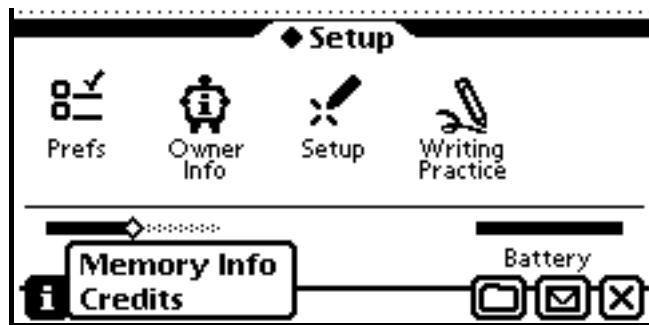
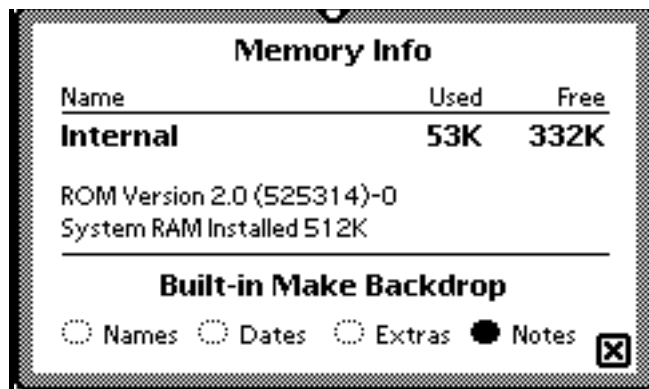
Getting Started

This chapter shows how to get started debugging your packages.

What You Need

You need the following to debug with Newtsbug:

- A Newton device using a 2.0 (or later) system. You should use a power adaptor rather than the Newton's battery, because the Newton will not sleep when stopped in the debugger.
- Newtsbug Connection, which comes on the Newtsbug CD, loaded on your Newton device.
- A Mac OS-based computer.
- On the Mac OS-based computer, an image file. The image file is a representation of the Newton system that can be used by Newtsbug. The image file you have needs to match the system on your Newton device. You can find out the system version number on your Newton device by, with the Extras drawer open, tapping on the memory info button inside the "i" button in the lower left corner. Figure 2-1 shows what the Memory Info command looks like; Figure 2-2 shows how the information appears. In this case, it is ROM 2.0 (525310)-0.

Figure 2-1 The Memory Info Command**Figure 2-2** Memory Info

- The Newtsbug application, loaded on the Mac OS-based computer.
- A serial cable.
- The package you want to debug loaded on the Newton. (You can load it later using Newtsbug, if you want.)
- The file with symbol information for the package in a place where Newtsbug can find it. If you used the Newton C++ Tools to create your package and you haven't moved the symbol file or package file, Newtsbug can find the symbol file through the package file. See "Files Newtsbug Needs to Find" on page 2-5 for information.

You may want to use a serial PCMCIA card, which leaves the Newton serial port free for other uses. We have tested the Socket Communications serial I/O card, and this works fine with current Newton devices; we recommend that developers use this. Other cards may work as well, but haven't been tested as of this writing.

Beginning Debugging

1. Create an alias for your package file and put it in the Newtsbug folder on your Mac OS-based computer.

Newtsbug looks for package files, or aliases to the package files, in its directory. It derives the expected name for a package from the Name entry of the Package Settings icon of the Project Settings menu item in NTK. If you suspect you don't have the right name, the easiest way to see the proper name is to start Newtsbug. If the name you have is wrong, you will see an error message that gives the name that Newtsbug expects. You can tell if your package symbols are being read successfully by watching the Newtsbug stdin/stdout/stderr window while you make the connection with the Newton. Newtsbug indicates when it successfully reads a package file and symbol file. See "Files Newtsbug Needs to Find" on page 2-5 for more information, particularly if you have moved the .pkg file or the file that contains the compiled C code, which ends in .sym. That section also has information on naming the package file so that Newtsbug can find it.

2. Run Newtsbug.
3. When Newtsbug asks for an image file, click on the Cancel button.
4. Choose the Preferences command from the Edit menu.
Newtsbug displays a dialog box that lets you set various features of Newtsbug.
5. The bottom part of the dialog box lets you select the modem port or printer port. Choose the one that you've used for the serial cable to the Newton.
6. Choose the baud rate you want. All of the available baud rates work well on most Mac OS-based computers. If you have a computer that has a processor that is slower than a 68040, choose something slower than 57600. If you have trouble maintaining a debug session, try lowering the baud rate.
7. If you expect to set breakpoints in ROM, step in ROM, or modify ROM, you need to reserve some memory using this dialog box. (If you are just going to take these actions in your package code, you do not need to reserve any memory; however, see note below.) See "Number of 4K Pages for Breakpoints" on page 3-7 for details. You can reset the amount of memory available by using the Preferences command again.

Note

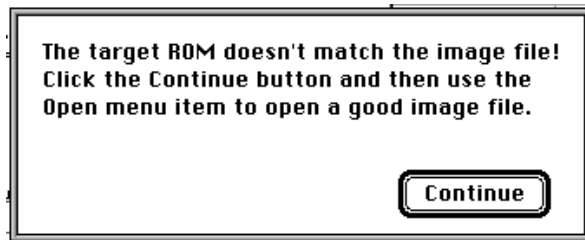
If you are using a Newton MessagePad 120 or 130 running Newton OS 2.0 and you want to use the C debugging function `printf`, you need to reserve two pages (8K), because those versions of the Newton system cannot otherwise execute `printf`. If you do not reserve space, calls to `printf` will do nothing. If you are using a later version of the Newton system, you do not need to reserve any space for this purpose. ♦

8. Click OK to close the dialog box.
9. Open an image file using the Open menu command in the File menu. Newtsbug reads the symbols from the file.

Getting Started

10. On the Newton, open the Newtsbug Connection application, which will be in the Extras drawer.
11. Select the same baud rate you chose in Newtsbug.
12. Make certain that Newtsbug shows the message “Ready for target to setup debugging from Printer/Modem port.” Do not proceed to the next step before that message shows. (The “target” referred to in this message is the Newton.)
13. Tap on Connect.

A NewtonScript Listener window should appear on the Mac OS-based computer. If the image does not match the image on the Newton, the debugger tells you that, showing a message like this:



You need to click on Continue and choose an image that matches the system on the Newton.

Newtsbug displays other error messages in the stdin/stdout/stderr window. You will probably see messages saying that Newtsbug could not find package files. Newtsbug looks in its folder for package files for all packages loaded on the Newton; you can ignore these messages for packages that you don't plan to debug.

Making Sure Your Package Symbols are Available

When you start a debug session and Enable Package Symbols (Config menu) is checked, Newtsbug scans the Newton device for currently loaded packages and looks for symbol files that are associated with the packages. Since the package you're interested in could have happened to have been swapped out of the Newton's memory, Newtsbug could fail to find information on your package. (You can ensure that this does not happen by opening the package right before you set up debugging.)

To figure out if your package's symbols are loaded, choose the Procedure command from the Procedures menu. That shows a list of all symbols that Newtsbug currently knows about. You can look at this list to see if the symbols from your package are loaded.

If they are not loaded, you can cure the problem by reinstalling the package on the Newton. When you do that, Newtsbug loads the package symbols.

Another possibility is that the package symbol file does not have the correct name. Look in the stdin/stdout/stderr window for error messages. If there is one that deals with loading package symbols, the error message includes the package name that Newtsbug expects. Change the name of the package file alias that is in the Newtsbug folder to match the name in the error message and reinstall the package.

Problem Alert Dialogs

When using Newtsbug, if a serious problem happens Newtsbug will put up a dialog describing the problem and allowing you to either quit immediately or continue. You get this kind of dialog, for example, when Newtsbug can't get memory to do something critical or can't communicate with the Newton. When you see that kind of dialog, you should consider quitting Newtsbug and judge whether or not you can continue based on what has happened on the Newton. If you decide to continue and then get more such dialogs, you should probably quit. For example, if you are developing a fairly large package, you might see Newtsbug complaining that it is unable to get memory for storing the package contents. You should quit Newtsbug, increase Newtsbug's application heap size, and try again. (You need at least 400K to debug packages.)

Ending a Debugging Session

To end a debug session:

1. On the Newton, open Newtsbug Connection again.
Now the button should say Disconnect.
2. Tap on the Disconnect button.

If you do not close a debugging session from Newtsbug Connection, calls to functions such as `DebugStr` and `printf` will cause exceptions on the Newton device, because the Newton will try to communicate with Newtsbug over the serial link.

Files Newtsbug Needs to Find

Newtsbug needs to be able to find the package file for your package. When you create a package that contains C code, the C part of the package comes from a `.sym` file that contains symbol information. Newtsbug also needs to be able to find that file in order to be able to debug the package.

Naming the Package File So Newtsbug Can Find It

Newtsbug uses a naming scheme to find the package file. It uses information in the package file to find the `.sym` file. The easiest way to find the name that Newtsbug wants is to start Newtsbug and look for the name given in the error message for that package. Newtsbug derives the name of the package file using these rules:

- If the package name has any colons (:), they are replaced with periods (.), because colons have a special meaning for the Mac OS file system.

Getting Started

- If the package name is longer than 24 characters, the name is truncated to 24 characters.
- If the name contains any periods (.), the last period and any characters after it are removed
- The characters .pkg are appended to the name.

Handling the .sym File

A package that contains C++ code includes a part information field that records the path from the package file to the .sym file. If you change the relative path that leads from the package file to the .sym file, Newtsbug will not be able to debug the package. For example, the Newton C++ Tools build system places the .sym file in a folder called Objects that is in the directory that contains the package file. If you move the package file, its new directory must have an Objects folder that contains the .sym file.

If You Aren't Getting Symbols For Your Package

If you've done everything discussed in the previous two sections and you still aren't getting symbol information, it is possible that the header part of the package was not in the Newton's main memory when you started the debugging session. To cure this problem, you can try the following tactics. Either of them may work; try whichever is easiest for you first.

- Re-load the package onto the Newton using Newtsbug.
- Open the application contained in the package on the Newton before connecting with Newtsbug.

Newtsbug also displays the error message "package header not currently in memory," but if you have more than one package loaded on your Newton, you cannot tell if the message is for the package in which you are interested.

Newtsbug Menus

This chapter describes the Newtsbug menus. Figure 3-1 shows the menu bar.

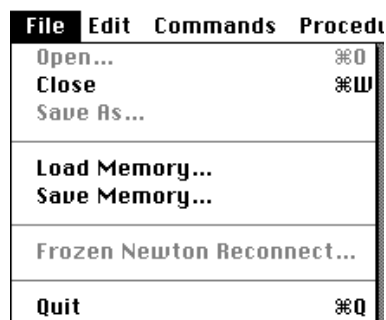
Figure 3-1 Menu Bar



File Menu

The File menu lets you open image files, close windows, save the contents of certain Newtsbug windows, and store and re-load memory.

Figure 3-2 The File Menu

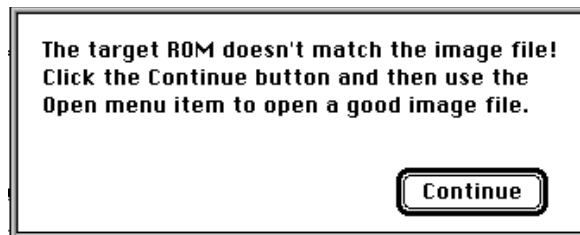


Open (⌘-O)

The Open menu item lets you select an image file and get Newsbug ready to accept a debug session from the Newton. An image file and its accompanying “high” file contain a version of the Newton ROM that Newsbug can load into itself and use. The image file you use must match the version of the Newton system that is on your Newton.

The debugger does not actually check if the image can be used with the Newton until you try to set up the debugging session from the Newton using Newsbug Connection. At that point, if the image does not match, Newsbug shows you the error message in Figure 3-3

Figure 3-3 Mismatched Image Error



Close (⌘-W)

The Close menu item closes the active window.

Save As...

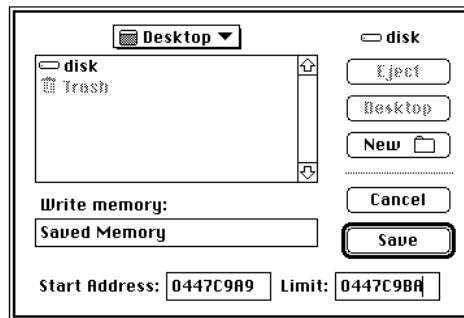
The Save As menu item lets you save, as text files, the contents of the I/O windows such as the Listener, stdin/stdout/stderr window, and heap windows.

Load Memory...

The Load Memory menu item lets you fill a piece of memory with a file that has been saved with the Save Memory command.

Save Memory...

The Save Memory menu item lets you save the specified memory, as raw binary data, to a file.

Figure 3-4 The Save Memory Dialog

The limit field is the address up to which, but not including, you wish to write. In the above illustration, memory from 0447C9A9 through 0447C9B9 will be written.

Frozen Newton Reconnect...

You can use this command to help track down unusual bugs that cause the Newton to freeze at unpredictable times. Here's the typical process of using this feature:

1. Set up a Newtsbug session as usual.
2. Disconnect the serial cable (make sure the Newton is running at the time).
3. On the Mac OS computer, select Quit from Newtsbug.
Newtsbug puts up a dialog saying that the current debug session is still on.
4. Click on the Quit Now button

Now the Newton has the debugger enabled, but still functions as a normal Newton. You can use the Newton normally, except don't use the serial port that was used in step 1 for setting up the Newtsbug session.

If the Newton crashes for a cause that can't be captured by Newton's exception handling process, it will be captured by the low level debugger. Examples are a bad PC value or a `DebugStr` call.

IMPORTANT

When this happens, the Newton will not power off. It just loops in a low level debug loop waiting for Newtsbug to connect to it. The battery may be used up very quickly. ▲

5. When the Newton freezes, connect it to a Mac OS computer with a serial cable; launch Newtsbug, open a proper image, and use the Frozen Newton Reconnect command.
6. If you can successfully connect, Newtsbug displays the current PC and the CPU window. You can do further debugging by opening stack window, memory windows, and so on. You can also set breakpoints or step in packages; although you can't do that in ROM. Some features such as printing of `printf`'s and use of the Listener window, are not supported in this mode.

7. After finishing debugging, you should reset the Newton, because the Newton's internal state may be bad.

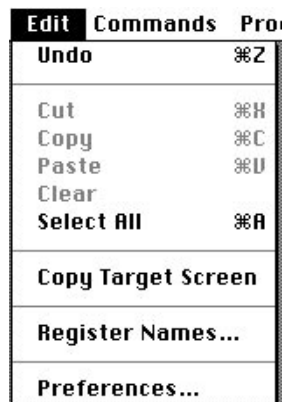
Quit (⌘-Q)

The Quit menu item quits Newsbug and ends this debugging session. If the debugging session is still open, Newsbug warns you and gives you a chance to end the debugging session from the Newton first. You should end the debugging session from the Newton before quitting Newsbug, unless you want to try to debug a Newton that is freezing. (See “Frozen Newton Reconnect...” on page 3-3.)

Edit Menu

The edit menu lets you use the standard Mac OS edit functions, Undo, Cut, Copy, Paste, Clear, and Select All, and also has various Newsbug functions. The standard commands are not documented in this manual; they work as they usually do in Mac OS programs.

Figure 3-5 Edit Menu



Edit	Commands	Pro
Undo		⌘Z
Cut		⌘H
Copy		⌘C
Paste		⌘U
Clear		
Select All		⌘A
Copy Target Screen		
Register Names...		
Preferences...		

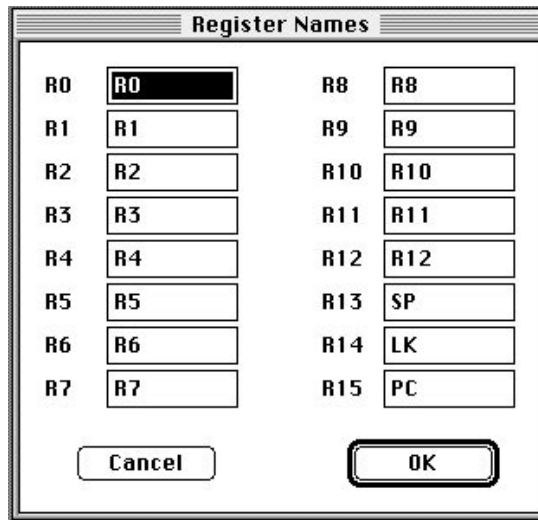
Copy Target Screen

The Copy Target Screen menu item copies the screen of the Newton device to the clipboard.

Register Names...

The Register Names menu item puts up a dialog that lets you name the registers, if you want to for some reason. See an assembler instruction manual for an explanation of the standard names. Figure 3-6 shows the dialog.

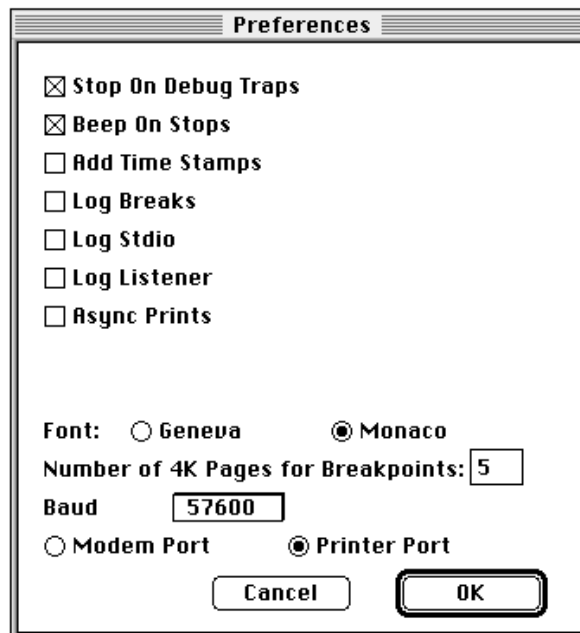
Figure 3-6 Register Names Dialog



Preferences...

The Preferences menu item puts up a dialog that lets you set various features of Newsbug. Figure 3-7 shows the dialog. The options are described below the figure.

Figure 3-7 Preferences Dialog



Stop on Debug traps

If this option is not checked, calls to `DebugStr` and `Debugger` do not stop, but `DebugStr` still displays its string.

Beep on Stops

This option enables and disables the audible indication of stopping.

Add Time Stamps

This option adds a time stamp to each line in the `stdin/stdout` window, output from the Newton in the Listener window, and to log files, using the Mac OS-based computer to determine the time.

The times given are not exact because there is a delay between when the image sends a string to the window and when Newtsbug gets it, and the delay cannot be calculated because it depends on other applications that may be running on the Mac OS-based computer.

Log Breaks

This option tells Newtsbug to log information in the `stdin/stdout/stderr` window each time any breakpoint is hit. This may be useful to trace the program flow, especially if you turn on Add Time Stamp option, as well. Newtsbug will display information similar to this:

```
2/10 break at PC = 00137FE8 TTracer::TTracer(char*) + 12
```

The first value, 2 in this example, is the current number of “hits” on the breakpoint. (Actually, the number shown is the number that appears in the Breakpoint Conditions Hits box; see “The Breakpoint Conditions Dialog” on page 5-2 for details.) The second value, 10 in this example, is the number in the “Stop After” box in the Breakpoint Conditions dialog.

If a breakpoint is conditional, or the “Stop After” checkbox in the Breakpoint Conditions dialog is not checked, information is logged even though the program does not stop at the breakpoint.

If you want the information written to a file, also turn Log Stdio on.

Note that turning on this option can slow the Newton significantly.

Log Stdio

This option tells Newtsbug to save everything in the `stdin/stdout/stderr` window to a log file. The log file is placed in the Newtsbug folder and is named “HLog stdio”. If a log file already exists when Newtsbug tries to open or create one, it is overwritten, so rename the file if you want to keep it. The log file is closed when you quit Newtsbug or you turn this option off, but not when you rerun.

Log Listener

This option tells Newtsbug to save everything in the NewtonScript Listener window to a log file. The log file is placed in the Newtsbug folder and is named "HLog listener". If a log file already exists when Newtsbug tries to open or create one, it is overwritten, so rename the file if you want to keep it. The log file is closed when you quit Newtsbug or you turn this option off, but not when you rerun.

Async Prints

This option tells Newtsbug to buffer printing to the stdio window and the Listener window as quickly as it can. Display of information in this window is typically delayed, especially when the image prints a lot. When you check this command, the image is not slowed to wait for window display.

Font

This option lets you set the font used in Newtsbug in the stdin/stdout/stderr and Listener windows to either Geneva or Monoco.

Number of 4K Pages for Breakpoints

You generally only need to reserve space if you are going to be setting breakpoints or doing other debugging actions in ROM (however, see the note following). You don't have to reserve anything for debugging your package code.

This option lets you set how many four-kilobyte memory pages are reserved for ROM breakpoints. If you expect to set breakpoints in ROM, step in ROM, or modify ROM, reserve a few pages. More memory allows more breakpoints. If you're going to do any of these operations, you need to reserve at least two pages, one for the system and one for stepping or the first breakpoint. For each breakpoint in a different ROM page, you need to reserve one additional page.

For example, suppose you set breakpoints at the following addresses (for your information, 4K is 0x1000 in hexadecimal representation):

```
0x10000
0x10020
0x10070
```

Those addresses are on the same page, so you need to reserve two pages: one for the system and one for the page that contains these addresses. Suppose you add a breakpoint at this address:

```
0x13004
```

That address is on a different page, so you need one additional page, for a total of three pages.

Now, suppose you want to add breakpoints at these addresses:

Newsbug Menu

0x20030

0x20938

Those addresses are on the same page, so you need one additional page, for a total of four pages.

If you try to set more breakpoints than those reserved memory can support, Newsbug informs you.

Note that the Newton has a limited amount of system memory, and you should not reserve more space than you need. On a MessagePad 120, you cannot reserve more than twenty pages, and generally should reserve less.

Note

If you are using a Newton MessagePad 120 or 130 running Newton 2.0 and you want to use the C debugging function `printf`, you need to reserve two pages (8K), because this version of the Newton system cannot otherwise execute `printf`. If you do not reserve space, calls to `printf` will do nothing. If you are using a later version of the Newton system, you do not need to reserve any space for this purpose. ♦

Baud Rate

This option lets you set the baud rate for communication with the Newton device. All of the available baud rates work well on most Mac OS-based computers. If you have a computer that has a processor that is slower than a 68040, choose something slower than 57600.

Modem Port or Printer Port

This option lets you choose which port you are using to communicate with the Newton device.

Commands Menu

The Commands menu has various useful commands.

Figure 3-8 Commands Menu

Commands	Procedures	Memor
Go		⌘G
Stop		⌘.
Rerun		⌘R
Clear All Breakpoints		
Clear All Breakpoint Hits		
Disable Breakpoints		
Step Into		⌘S
Step Over		⌘T
Convert...		⌘N
Download Package...		⌘2
printdepth := \		⌘3
breakonthrows := \		⌘4
DU(Debug("\		⌘5
stacktrace()		⌘6
exitbreakloop()		⌘7
dv('viewfrontmost)		⌘8
getview('viewfrontmost)\		⌘9

Go (⌘-G)

The Go command causes a stopped image to resume execution.

Stop (⌘-.)

The Stop command suspends execution of the image currently being executed.

The image might not stop if:

- Interrupts are disabled (Newtsbug uses an IRQ interrupt to stop and start the image)
- The image is too lost to respond to the interrupt because of, for example, bad MMU tables
- There are hardware problems such as loose or bad cables

Rerun (⌘-R)

This menu item is similar to using the reset button on the Newton. (The primary difference is that if you use the rest button, Newtsbug will not know that the Newton has been reset, and Newtsbug will not reset its state.) All breakpoints are removed. All changes in code are lost. You will need to re-establish the debug connection.

Clear All Breakpoints

This menu item clears all breakpoints, both temporary and permanent. (Breakpoints are discussed in “Breakpoints” on page 5-1.)

Clear All Breakpoint Hits

You can set breakpoints so that they trigger after a certain number of hits. This menu item clears all breakpoint hits. See “Breakpoints” on page 5-1 for more information on breakpoint hits.

Disable/Enable Breakpoints

This command toggles between disabling and enabling breakpoints. When breakpoints are disabled, they will never trigger.

Step Into (⌘-S), Step Over (⌘-T)

If the instruction at the current PC is a BL instruction, the menu items show as Step Into and Step Over. Otherwise they are simply Step.

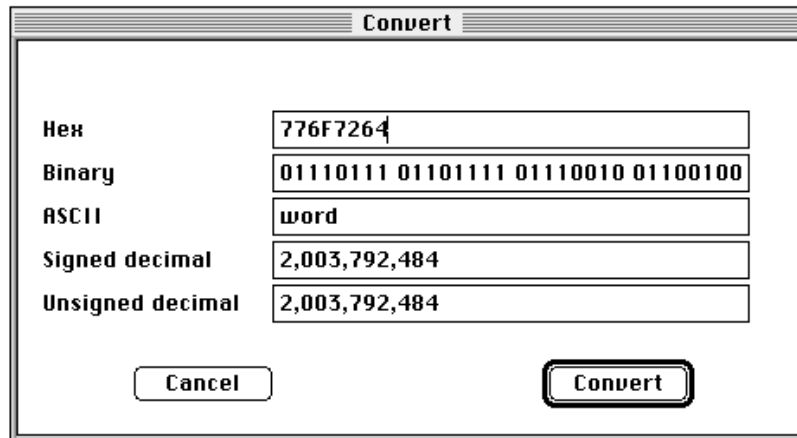
These commands are used to advance program execution one instruction, or step, at a time. They operate in the same way as the Step/Step and Step Into/Step Over buttons provided in Newtsbug’s Status window (described in “Go and Step Buttons” on page 4-2). They simply cause the next instruction to be executed or, if the instruction is a subroutine call, let you either step into the subroutine or step over the calling instruction.

Warning

Do not step in the `MainSCCInterrupt` handler because Newtsbug will either lose the connection with the Newton device or crash. In general, you should not step through system code. If you do so, you may need to do a cold re-boot of the Newton device. ▲

Convert... (⌘-N)

The Convert command displays a simple calculator. Given a number in hex, binary, signed or unsigned decimal, or a sequence of characters in ASCII, all other formats are displayed. If an 8-digit hex value is selected before Convert is chosen from the menu, this value is displayed in all possible formats. You change the value in any format.

Figure 3-9 Convert Dialog

If any of the values does not have an ASCII representation, the ASCII field shows a small box in place of each character that can't be represented.

Download Package...

This command puts up a standard file dialog and lets you choose a package to download. It then loads the package on the Newton device, first removing any old version of the package that is on the Newton device.

▲ WARNING

You cannot delete a package on the Newton device while the Download Package dialog is displayed. If you try to do so, the Newton device will claim to be deleting the selected icons, but the operation will never complete. If you cancel the download package command quickly, the Newton operation will fail. If you wait too long, however, the Newton device may hang. ▲

Command Keys for the Listener

<code>printdepth := \</code>	⌘3
<code>breakonthrows := \</code>	⌘4
<code>DU(Debug("\</code>	⌘5
<code>stacktrace()</code>	⌘6
<code>exitbreakloop()</code>	⌘7
<code>dv('viewfrontmost)</code>	⌘8
<code>getview('viewfrontmost)\</code>	⌘9

The last group of commands in the Commands menu are command keys for the Listener window.

Newtsbug will type the characters in the command you select in the current line in the Listener window. If the last character in the string is not a back slash (\), Newtsbug,

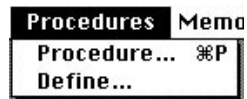
enters the command, and executes it as if you have typed the command and pressed the enter key. If the last character in the string is a back slash, Newtsbug simply puts the string to the Listener window without executing it. You can then edit the text or add to it and press enter to execute it.

You can customize all of these command keys. To customize, create a text file named Listener Commands in Newtsbug's folder. Each line of the file becomes a command in the Commands menu. See "Customizing NewtonScript Listener Shortcuts" on page 5-5 for details.

Procedures Menu

This menu is used to find or define procedures so that you can quickly access them using code windows. As shown in the figure, this menu usually includes some code window names, which are listed below a line under the Define menu item. See "Code Windows" on page 4-6 for information on those windows.

Figure 3-10 Procedures Menu



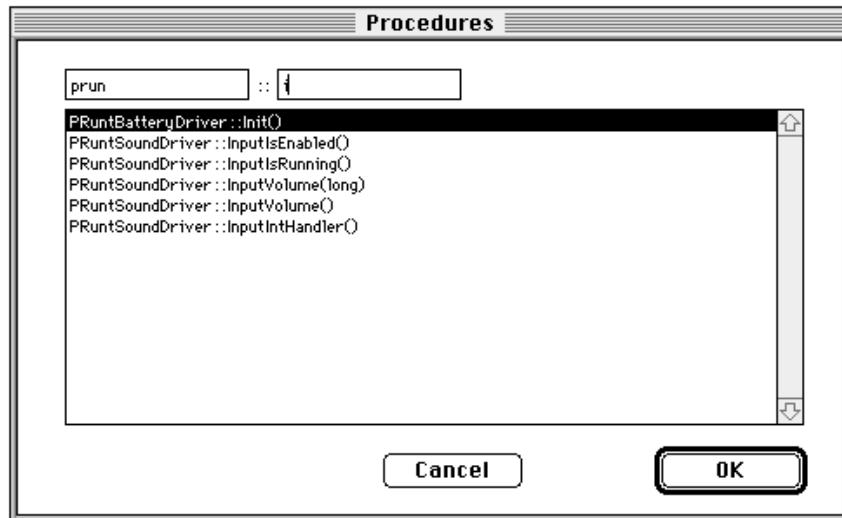
Procedure... (⌘-P)

This item opens a symbol browser. You can type a class prefix in the class box and a member prefix in the member box. All matching procedures are listed. When you click "OK," a window showing the procedure is opened.

Newtsbug's search for a symbol is not case sensitive and a prefix of a symbol can be used (for example, "asyncc" to find `ASyncCallback`).

For convenience, global names are accepted in the class box.

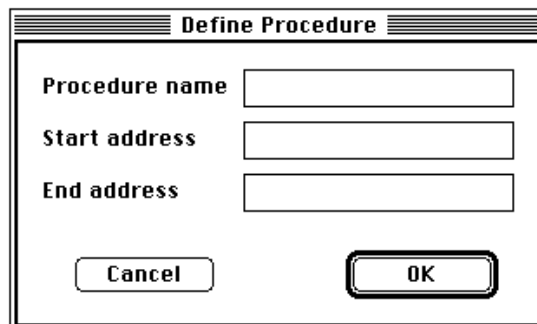
Figure 3-11 Procedure Browser



Define...

With this menu item, you can temporarily define a procedure name to reference a particular range. These definitions are remembered for the duration of the current debugging session but not from session to session.

Figure 3-12 Define Procedures Dialog



Memory Menu

The Memory menu lets you view and change the memory state and contents.

Figure 3-13 Memory Menu



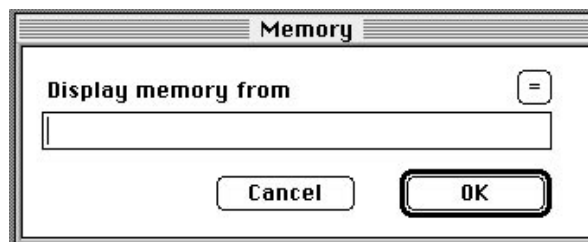
Memory...

Memory windows let you display and modify the contents of RAM or ROM. Selecting this menu item while a value is selected in a Newtsbug window opens a memory window at the selected address. If no value is currently selected, Newtsbug brings up a dialog requesting an address to display. Figure 3-14 shows the dialog. You can enter:

- An address in hex
- A symbol such as a global variable name
- An expression; see “Expressions” on page 5-4 for details

You can press the = button to evaluate the expression

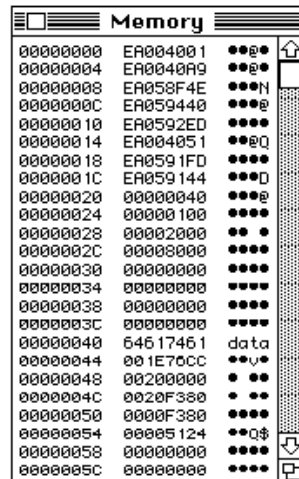
Figure 3-14 Memory Dialog



Newsbug Menus

Once an address is entered, a memory window is opened. The three columns displayed are address, hex value and ASCII value. This window can be scrolled $\pm 4K$.

Figure 3-15 Memory Window



If you hold down shift when opening a memory window while an address is selected, Newsbug treats the address as a handle instead of a pointer.

The hex values displayed in the window can be edited. To modify the content of an address, simply select all or part of the hex value and edit it. Use the mouse to move the cursor to another place when you are done. Currently the ASCII value cannot be edited.

Frame...

This menu item lets you display the specified variable or selection as a NewtonScript object.

If you have a number selected when you choose this command, what happens depends on what the number is and whether or not you have the Shift key down. Table 3-1 shows the various possible results.

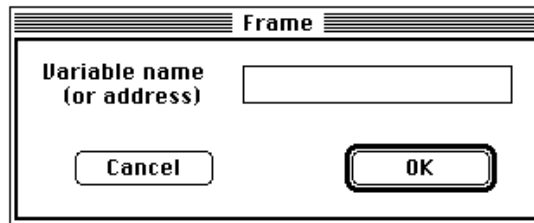
Table 3-1 The Frame Command

Selected Number	Effect With Shift Key Up	Effect With Shift Key Down
The address of a frame or array	Opens a window displaying the frame or array	Don't do this ¹
A Ref ² of a forwarding object that forwards, eventually, to a frame or array		
The address of a forwarding object that forwards, eventually, to a frame or array		
A magic pointer Ref referring to a frame or array		
A Ref of a frame or array	The number is treated as an address, and a memory window is opened at the selected address	Opens a window displaying the frame or array
The address of a Ref of a frame or array		
The address of an address of a Ref of a frame or array		
A Ref of a forwarding object that forwards, eventually, to a frame or array		
The address of a Ref of a forwarding object that forwards, eventually, to a frame or array		
The address of an address of a Ref of a forwarding object that forwards, eventually, to a frame or array		
A magic pointer Ref referring to a frame or array		
The address of a magic pointer Ref referring to a frame or array		
The address of an address of a magic pointer Ref referring to a frame or array		
Any other number		

¹ With the shift key, down the selected number cannot be the address of an object—something will happen but not what you want. Holding down shift is useful if the number selected is the address or contents of a RefVar, RefStruct or RefHandle.

² A Ref is the NewtonScript implementation's reference to a NewtonScript object (a NewtonScript pointer, if you will.)

If nothing is selected when you choose this menu item, Newtsbug displays the dialog box shown in Figure 3-16.

Figure 3-16 Frame Dialog

If you enter a number the behavior is as if the number had been selected, although the Shift key is ignored. If you enter a symbol or the prefix of a symbol, then the behavior is as if the symbol's address had been selected and the shift key was down.

Figure 3-17 shows a frame window for gVarFrame.

Figure 3-17 Frame Window

Variable Name	Address	Size	Value
Address: 0440D950 Size: 29 Class: 'Frame'			
classes	044045B9	Frame	{12}
extras	04417489	Array	[10]
navigator	04417279	Frame	{1}
AvailablePrinte...	04417591	Array	[5]
international	04417401	fFrame	{2}
routing	04417119	Frame	{1}
poweroffhandlers	04417681	Array	[0]
psFonts	04417041	Frame	{1}
userConfigurati...	04418179	f<Fault Block>	[4]
vars	044045F9	fFrame	{29}
actionDescripti...	FFFF8164		-8103
formulaList	04417911	Array	[4]
displayParams	044170A9	fFrame	{2}
notifications	04417611	Array	[1]
soupNotify	044177B1	Array	[10]
fonts	04416FD9	Frame	{1}
dictionaries	04418701	Array	[25]
preferences	04417859	Array	[14]
cardFilePrefs	04417379	Frame	{1}
cardSoups	044176F9	Array	[16]
paperPrefs	044172F9	Frame	{1}
setTimeSeed	00000000		0
functions	04404FB9	fFrame	{632}
trace	00000002		NIL

The area of the Frame window below the header includes the address of the object header, the number of slots, the class and the Dirty, Writeable and Locked flags as icons. You can select the address of the object header.

You can get additional information by clicking on parts of the frame window.

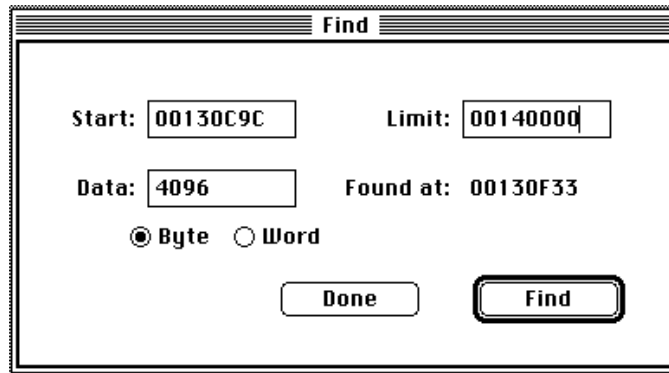
- Clicking on the Ref in the middle column selects it like any other Newtsbug number.
- Clicking on a tag in the left column or the value in the right columns is the same as selecting the Ref and invoking the Frame command with the shift key up; that is, it opens a Frame or memory window on the object in that slot.

- Clicking on a number slot will bring up the Convert window, showing the number in various formats.

Find...

Find searches for the specified data, as a word or a byte, in the range specified. The limit value is the address up to which, but not including, you wish to search.

Figure 3-18 Find Dialog



You can click the Find button successively to continue searching. When no new match is found, the “Found at” field says “Reached End.” To start the search over, click the Find button again.

Heap

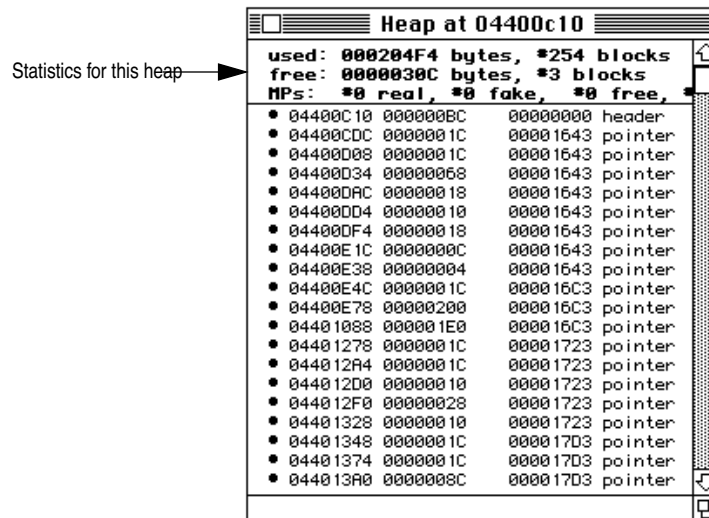
Selecting the Heap item displays four windows showing four different heaps. The topmost window displays the contents of the pointer heap, indicating the memory blocks returned by `malloc` and `NewPtr` calls. The handle heap shows block returned by `NewHandle`. The master pointer heap (mps) shows the master pointers used for the handles, and the wired heap is used by the operating system.

If an 8-digit hex number is selected, Heap opens a single heap window using the selection as the address of the heap header.

Note

Newtsbug can slow dramatically if you leave heap windows, including the script heap window, open while you step. This is because each time the Newton is stopped, Newtsbug updates every open window. Since heap windows reference a lot of memory information, the updating process can take a significant amount of time. You should therefore close unnecessary heap windows before you step or let the Newton go. ♦

Figure 3-19 Heap Window



Heap windows can be sorted by address, logical block size and task ID, by choosing the Sort by Address, Sort by Size, and Sort by Task menu items respectively.

Script Heap

This item displays a window showing the script heap.

Objects that are "dirty" and can be garbage collected are indicated by a dot.

Smash Heap Tags

Smash Heap Tags sets all tags of the active heap to 0x7fffffff. It cannot be undone. This is useful to see new heap allocations after this action. This command is only active when the heap window is in front. (Note that this command can take a few minutes to execute.)

Sort by Address

Sorts the items in the selected heap window by address. This command is only active when a heap window is in front.

Sort by Size

Sorts the items in the selected heap window by size. This command is only active when a heap window is in front.

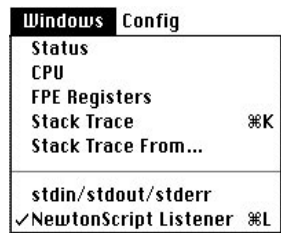
Sort by Task

Sorts the items in the selected heap window by task. This command is only active when a heap window is in front.

Windows Menu

The commands in this menu let you display the various windows that Newtsbug provides, except for code windows, which are displayed in the Procedures menu. The window that is currently “in front” has a check mark next to it. (If no window has a check mark, then the frontmost window is probably one of the code windows.)

Figure 3-20 Windows Menu



Status

This menu item brings the Status window to the front. See “Status Window” on page 4-1 for more information.

CPU

This menu item brings the CPU window to the front. The CPU window shows the state of the CPU. See “CPU Window” on page 4-3 for more information.

The FPE Registers

This menu item displays the FPE Registers window or brings it to the front. This window displays the contents of the floating-point registers. They cannot be edited.

Stack Trace

Selecting the Stack Trace item from the Windows menu displays the call chain of procedures to this point. See “Stack Trace Window” on page 4-5 for more information.

Stack Trace From...

This menu item lets you specify the point at which the stack trace should begin.

stdin/stdout/stderr

This menu item brings the stdin/stdout/stderr window to the front. See “Stdin/Stdout/Stderr Window” on page 4-6 for more information.

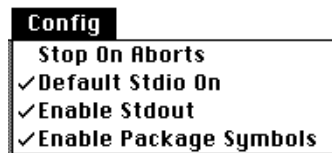
NewtonScript Listener

This menu item brings the Listener window to the front. See “NewtonScript Listener Window” on page 4-6 for more information.

Config Menu

The Config menu lets you use Newtsbug to configure images in various ways.

Figure 3-21 Config Menu



Stop on Aborts

Having this option on is useful for finding certain kinds of bugs. For example, if the program accesses an invalid memory address, there will be a data abort. If you have this setting on, the Newton stops when this problem happens, allowing you to examine the current PC and the stack.

Default Stdio On

This option has no effect.

Enable Stdout

If your C code has `printf` statements, you need to turn this item on so that the print can be shown in the stdin/stdout/stderr window. If you don't have this setting on, the `printfs` are equivalent to no-op.

Newsbug Menus

`DebugStr` and `DebugCStr` output will always be displayed regardless of the `stdout` setting.

Enable Package Symbols

This option is initially set, and causes you to see your symbols loaded from a package.

When you have this set, you get a warning message every time Newsbug finds a package on the Newton that does not have a symbol file on the Mac OS-based computer. You can ignore these messages—the only package that needs to have a symbol file is the one you want to debug. Turn this item off if you are not debugging a package. (The setting is “remembered” in the Newsbug preferences file.)

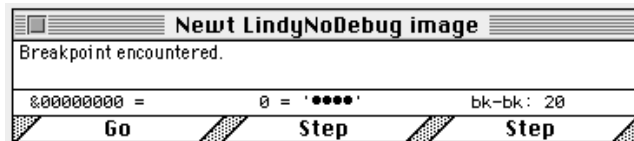
Newtsbug Windows

This chapter describes the Newtsbug windows.

Status Window

The Status window has three parts.

Figure 4-1 Status Window



The top part holds a message that tells you the status of the debugger.

The middle portion has a hex converter (see “Handy Hex Converter” on page 4-3).

The bottom shows one, two, or three buttons depending on the state of your program. When the Status window is first opened (when the image is launched) the window displays just one button at the bottom of the window, which can change depending on the state of the program:

- If you haven’t yet selected an image, the button says Open Image, and when you click on it you get a standard file dialog box that lets you choose an image.
- If you’ve chosen an image, but haven’t yet connected to a Newton, or you’ve closed the connection to the Newton, the button says Close Port. If you click on it, the serial port is closed.
- If you have closed the port, the button says Open Port. If you click on it, the serial port is opened.

- If you've chosen image, the button says Initializing while Newtsbug is opening the image or connecting with the Newton. The button does nothing at this point.
- If you are connected to a Newton and the image is running, and the button says Stop.
- If you click on the Stop button, Newtsbug stops the image and the Newton, and you enter the debugger. The button says Go. If you click on the Go button, the image starts again.
- If the image stops because of a breakpoint or error, the Go button and two Step buttons appear. They are described in the next section.

Go and Step Buttons

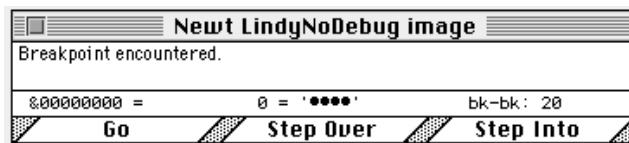
When you enter the debugger because a breakpoint or error was encountered, three buttons labeled Go, Step, and Step are displayed as shown below.

The Step buttons are just like the Step (Into) and Step (Over) menu items. They both cause the next instruction to be executed. If the next instruction is a subroutine call, the buttons change to Step Over and Step Into, allowing you to either step over the calling instruction or step into the subroutine.

Warning

Do not step in the MainSCCInterrupt handler because the debugger will either lose the connection with the Newton or crash. In general, you should not step through system code. If you do so, you may need to do a cold re-boot of the Newton device. ▲

Figure 4-2 Status Window Showing Step Over



NOTE

Breakpoints are not installed during Step or Step (Into) operations. Breakpoints are installed when you Go or Step (Over). ♦

Rerunning

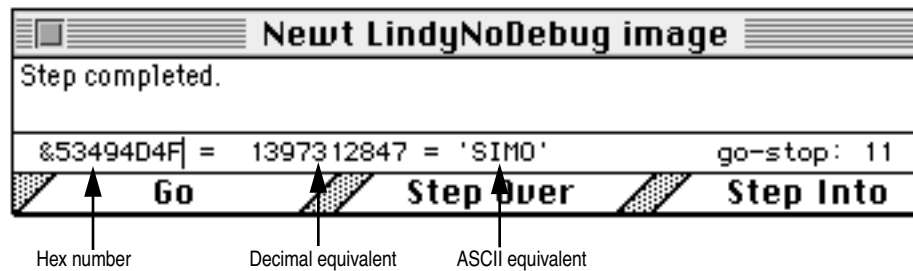
When your program completes, the buttons are labelled Quit and Rerun. Note that the buttons have command key equivalents as indicated on the Commands menu. (The command key equivalents are the same as their Macsbug counterparts.)

If the Newton has crashed, you need to press the Reset button to rerun.

Handy Hex Converter

The section of the Status window above the buttons has a hex converter that shows a hex value and its decimal and ASCII equivalents. The hex field shows the most recent value selected, but you can edit it to put in any number you want. The hex value can also be selected and copied.

Figure 4-3 Hex Converter



CPU Window

The CPU window has panes for the PC and PSR, the four banks of ARM General registers, the Timer register, and the MMU registers. Clicking on a pane label (MMU for example) toggles the state of the pane, opening it if it is closed and vice versa. The PC/PSR pane is always visible.

You should keep the panes closed when you aren't using them, because when they are open it is fairly easy to accidentally select fields in the panes, and if you select certain fields and then execute code, the results are unpredictable.

Figure 4-4 CPU Windows

CPU		CPU	
PC	0005724C	PC	000571BC
i f n z C v usr		i f n z C v usr	
General		General	
R0	00000000	R0	04007624
R1	00002710	R1	00002710
R2	00000000	R2	0400760C
R3	00001388	R3	00000000
R4	0400760C	R4	00001388
R5	00000093	R5	000013FC
R6	00000000	R6	0410123C
R7	00000000	R7	04101240
R8	00000000	R8	04007604
R9	00000001	R9	041013B8
R10	0410140C	R10	0410140C
R11	04007608	R11	04007608
R12	40000000	R12	0400760C
SP	040075E0	SP	040075E0
LK	00000000	LK	0021D198
Supervisor		Supervisor	
FIQ		FIQ	
IRQ		IRQ	
ABT		ABT	
Timer		Timer	
MMU		MMU	
		Base 01000000	
		Envir 55555555	
		Fault Bus Err 8	
		Domain 0	
		Addr 033C000C	

The PC/PSR pane shows the current value of the PC and a symbolic interpretation of the current PSR bits. You can change the PC. Clicking on the PC label is a quick way to find the PC. If you have opened several windows or scrolled the window containing the PC, clicking here will ensure that the PC arrow is visible.

The line under the PC is the current processor status register (CPSR). Clicking on the mode will cycle through the processor modes *usr*, *fiq*, *irq*, *svc*, *abt*, and *und*. \mathbb{H} -clicking on the mode will toggle between the corresponding 32-bit and 26-bit modes (such as *usr* and *u26*; *abt* and *und* have no 26-bit equivalent.). The *und* mod registers are not shown because they are used by Newtsbug.

The status register at the bottom of the Supervisor, FIQ, IRQ, and ABT panes is the saved processor status register (SPSR) for that mode. It is where the CPSR is saved by the processor when that mode is entered. SPSRs can be changed in the same way the CPSR can be.

Clicking on a PSR flag toggles its value; upper case indicates the flag is set, lowercase indicates clear. The I (IRQ) and F (FIQ) flags operate in the reverse: I or F indicates that the interrupt is disabled, i or f indicates that the interrupt is allowed.

Clicking on a register name opens a memory window starting at that register's value.

Clicking on the value of a register highlights that value for editing. The 8-digit hex number temporarily becomes a TextEdit field, complete with Cut, Copy, Paste and Undo. When you are finished editing the value, typing return replaces the original value with the new value.

The register bank labels (R8, LK, etc.) are displayed in black for the active bank and gray for inactive banks.

Note

Register names are grayed to indicate that they are not in the current CPU mode. You can change the values in the grayed registers, but you have to be very careful that you know the effect of the changes you make. ♦

FPE Registers Window

This window can be displayed by choosing the FPE Registers item from the Windows menu. It shows the contents of the floating-point registers. They cannot be edited.

Stack Trace Window

Selecting the Stack Trace item from the Windows menu displays the call chain of procedures to this point. The call chain is defined using R11 as the frame pointer. A nil value (zero) terminates the chain.

Clicking on a procedure name in this window opens the corresponding code window.

Pressing Option while clicking on a procedure name displays the stack frame of the procedure called by the selected procedure.

Pressing Command while clicking on a procedure name is equivalent to setting a temporary breakpoint and then giving a Go command, so the Newton runs until the point you've clicked is reached. (See "Breakpoints" on page 5-1.) You can use the Stack Trace From... menu item to specify the point at which the stack trace should begin.

When this window is in front and you select the Copy command from the Edit menu, Newtsbug copies the contents to the clipboard. (You can't select portions of the window contents, though.)

Stdin/Stdout/Stderr Window

This window shows error messages, `printfs`, and other program output. You can use the Edit menu commands, and you can use the Save As command to save its contents to a file (see “Save As...” on page 3-2). You can also set up the debugger so that it saves information to a file (see “Log Stdio” on page 3-6) and you can have output from the Newton time-stamped (see “Add Time Stamps” on page 3-6).

This window is not an editor window, and you cannot edit the contents using keyboard keys such as delete. (You can use the Edit menu Cut, Paste, and Clear commands, as well as Copy, though.)

NewtonScript Listener Window

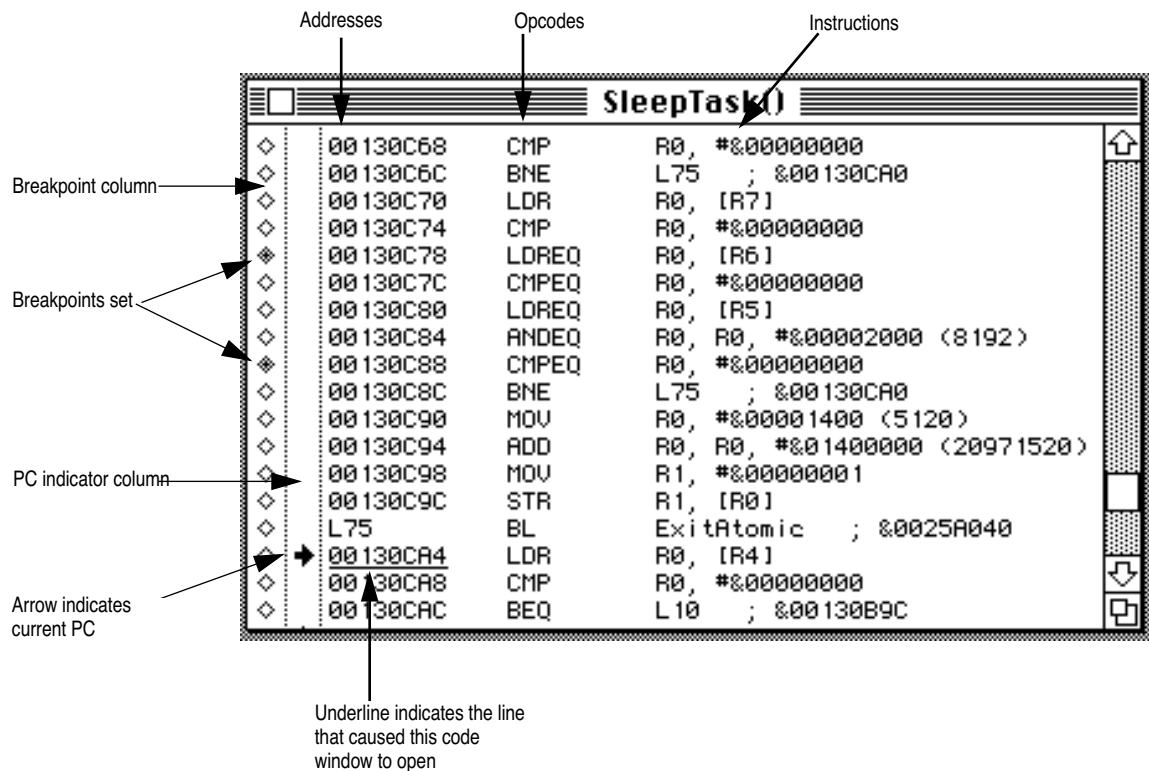
This window is similar to the NTK Inspector. You can type NewtonScript code. When you press the Enter key or ⌘-Return, the current line is transmitted to the Newton device for execution. You can also select one or more lines and press Enter or ⌘-Return.

In addition:

- You can use the Edit menu commands.
- You can save the contents to a file (see “Save As...” on page 3-2)
- You can log the contents to a file (see “Log Listener” on page 3-7)
- You can use items from the Commands menu to insert strings (see “Command Keys for the Listener” on page 3-11)
- You can have output from the Newton time-stamped (“Add Time Stamps” on page 3-6)
- There are a number of shortcuts (see “In the NewtonScript Listener Window” on page 5-5)

Code Windows

The Procedures menu includes a list of all open code windows. To display a code window, select it from the menu. In the menu, window names are displayed in the order in which the windows were opened (with the last one opened at the bottom) rather than in any calling or layer order.

Figure 4-5 Code Window Example

Each procedure is displayed as a separate code window. These windows are often opened automatically for you. For instance, when an exception or breakpoint is encountered, the code window containing the PC is opened and brought to the front. As another example, if you step into a procedure call, the called procedure's code window is opened. Currently Newtbug only knows about procedures with external linkage (that is, it doesn't know about static procedures). Static procedures get appended to the previous external procedure.

Opening Code Windows

You can manually open code windows in two ways:

- Click on a procedure name that is the operand of a branch instruction.
- Select Procedure from the Procedures menu and type in the name or address (in hex) in the Procedure Browser. Searching for a symbol name is not case sensitive and a prefix of a symbol can be used. For example, you could use `asyncc` to find `AsyncCallback`. See "Procedure Browser" on page 4-8 for more information.

Code Window Contents

Code windows display assembly language. Clicking in the opcode column beside an assembly language statement toggles the display of that line between disassembled text and the hex value of the instruction.

Clicking on the operand of a B or BL instruction does different things depending on whether the operand is a function or a label. If a function, Newtsbug displays a code window showing that function. If a label, Newtsbug shows that line. The PC does not change.

⌘-clicking on the opcode toggles between the original instruction and a NOP.

⌘-clicking in the PC indicator column sets the PC to the location where you click. (Be careful with this command, as you cannot undo it.)

Clicking in the PC indicator area is equivalent to setting a temporary breakpoint and then giving a Go command, so the Newton runs until the point you've clicked is reached. See "Breakpoints" on page 5-1.

Clicking on the operand area of a branch instruction shows a code window of the place the code branches to.

Code windows allow read-only selection of immediates and the address of PC-relative LDR/STRs.

The hex value is shown as a DCD directive, with ASCII in comments. The hex part of the directive is editable. Hex numbers are indicated with "&". Branch targets in the same procedure are shown as "Lnn" indicating the line number in the procedure that is the target of the branch.

You can toggle between a DCD directive and an assembly instruction by clicking on the opcode. You cannot change the instruction when it appears in assembly.

Changing the high nibble of a hex opcode to F will cause the instruction to never execute (such opcodes are actually reserved for other uses on future ARM processors). You can also ⌘-click on the opcode to change the instruction to a no-op. ⌘-clicking again changes the instruction back.

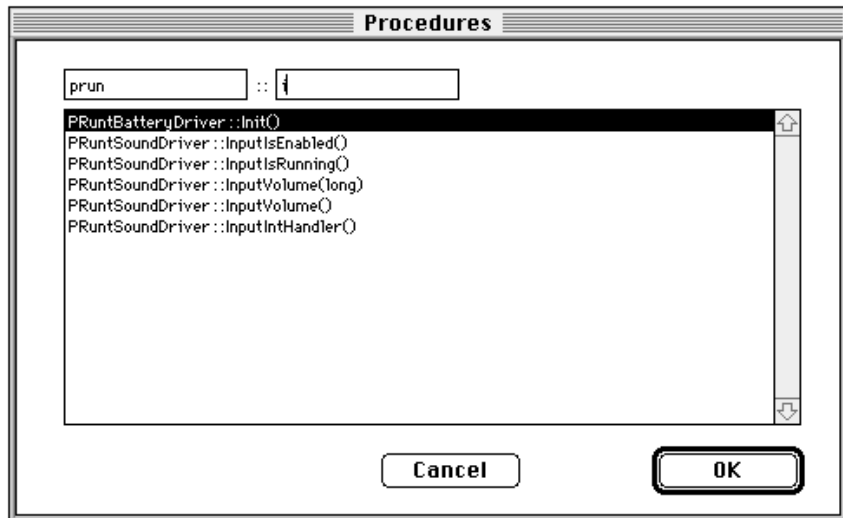
Unreachable instructions in C code are automatically shown in hex.

Procedure Browser

You can display the Procedure Browser choosing the Procedure command from the Procedures menu or by typing ⌘-p. This browser lets you see all the current procedure names or look at the member functions of a given class. You can type all or part of a class name or procedure name in the leftmost field at the top of the browser and all or part of a procedure name in the rightmost field.

Figure 4-6 shows the Procedure Browser, demonstrating how the browser sorts the existing names to match the partial names in the input fields.

Figure 4-6 Procedure Browser



Clicking OK or double-clicking on a function causes Newtsbug to open a code window for the highlighted function.

Newtsbug Windows

Using Newtsbug

This chapter has information on how to use Newtsbug.

Breakpoints

There are two basic kinds of breakpoints:

- Temporary breakpoints. You can set a breakpoint by clicking on a PC indicator column (the second column) in a code window. (See “Code Windows” on page 4-6.) The Newton stops briefly as the debugger sets a temporary breakpoint. The debugger then issues a Go command. When the temporary breakpoint is hit, the Newton stops and drops into the debugger, and the breakpoint is automatically cleared.
- Permanent breakpoints. You can set a permanent breakpoint by clicking in the diamond in the leftmost column of the code window. Permanent breakpoints are not cleared until you remove them by clicking again on the diamond, by using one of the Clear All Breakpoints command or the Rerun command in the Commands menu, or by quitting the debugger.

There are two kinds of permanent breakpoints:

- Unconditional breakpoints. This kind of breakpoint causes a break whenever it is encountered. An unconditional permanent breakpoint is indicated by a solid diamond.
- Conditional breakpoints. This kind of breakpoint causes a break only when a certain condition is met. Conditional statements (such as `BEQ`) automatically get conditional breakpoints that break only when the condition is met. You can see the Breakpoint Conditions dialog by \mathfrak{H} -clicking on the diamond; that dialog lets you control the breakpoint in a number of ways; see “The Breakpoint Conditions Dialog” on page 5-2 for details.

Warning

Do not set breakpoints in the MainSCCInterrupt handler because the debugger will either lose the connection with the Newton or crash. In general, you should not set breakpoints or step through system code. If you do so, you may need to do a cold re-boot of the Newton device. ▲

Note

You can also use debugger traps, which act in a way similar to breakpoints, but are faster. From C, you use the routines `Debugger` or `DebugStr` as a debugger trap. From assembler you use the `Debugger` or `DebugStr` macros. Unlike with Macsbug, the `DebugStr` parameter is a C string, not a Pascal string. ◆

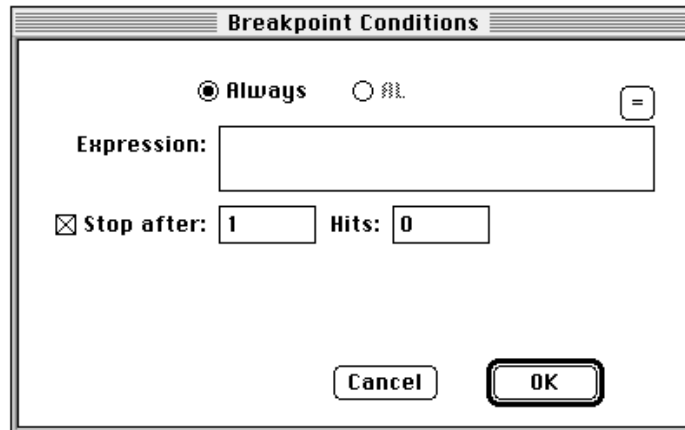
You can tell Newtsbug to log information whenever any breakpoint is hit. See “Log Breaks” on page 3-6 for information.

The Breakpoint Conditions Dialog

You can use the Breakpoint Conditions dialog to control what happens at a breakpoint in these ways:

- If the breakpoint is at a conditional statement, the break happens by default only when the condition is met. You can decide if you want the break to happen even if the condition is not met.
- You can have the break happen according to additional conditions so that, for example, the break only occurs if R0 has a particular value.
- You can have the break happen after the statement is hit a given number of times.
- You can set up the breakpoint so that the program never stops at it. (This is useful because you can have Newtsbug count the hits and also log information at the breakpoint.)

You can see the Breakpoint Conditions dialog by ⌘-clicking a breakpoint. It allows you to control the way the breakpoint works.

Figure 5-1 Breakpoint Conditions Dialog

It contains:

- A pair of radio buttons that let you determine what is defined as a “hit” in the case of a conditional statement, such as a BEQ. The leftmost one says “Always.” (For non-conditional statements, that is the only radio button that you can choose.) When you define a hit as Always, the statement counts as a hit whenever it is reached, even if the condition is not satisfied and the statement does not execute. The second radio button is for conditional statements, so that you can have the statement count as a hit only when the condition is met. For example, if the statement is a BLLE, that button will be labeled “LE”, and checking it means the statement will not count as a hit unless it actually branches. The default when clicking the breakpoint diamond of a conditional instruction is to have a conditional breakpoint.
- The Expression box, which you can use for a conditional expression. This lets you set an additional condition for counting the statement as hit. You use standard conditional expression syntax. For example, `r1==0`. If you want to test that you entered a valid expression, or see what the current state of the condition is, click on the “=” box, and Newtsg evaluates the expression.
Note that using the radio buttons for a conditional statement is much faster than using a conditional expression entered in the Expression box.
- A Stop After check box, which is followed by a box for entering a number. If the checkbox is checked, the breakpoint stops after the number of hits given in the number box. If the checkbox is not checked the program counts the hit, but doesn’t stop. You can set Newtsg to print log information every time a breakpoint is hit, whether or not the breakpoint is set to stop at that hit. See “Log Breaks” on page 3-6 for information.
- The Hits box. This shows the number of times the statement “hit” since you set the breakpoint. The maximum value is 32767. This value is not automatically reset after you “go” past the breakpoint, but you can edit it, if you want.

Expressions

You can use expressions in a number of places in Newtbug. The expressions are the same as C expressions with the following exceptions:

- The numeric radix is hex. To get a decimal number use a leading '#'. For example:
 - '10' is decimal sixteen
 - '#10' is decimal ten
- The names 'r0' to 'r15', 's1', 'fp', 'sp', 'ip', 'lr', 'lk', 'pc', 'cpsr', and 'spsr' refer to the contents of the corresponding registers of the current mode. They are considered to be unsigned ints.
- You are allowed to indirect through integers as if they were cast to `int*`, thus `*4` means `*(int*)4`
- The names of non-static, file-scope variables are recognized and they are interpreted as addresses. Therefore, to examine the value of a global `gX` use
 - `*gX` if it is `int` or `long`
 - use `(*gX >> #16) & 0xffff` if it is `short`
 - use `(*gX >> #24) & 0xff` if it is an unsigned `char`.
 The parentheses in these examples are actually redundant given C's operator precedence.
- No user-defined types or typedefs are recognized so the dot operator (`.`), `->`, and `[]` are useless.
- The various assignment operators (`=`, `+=`, `++`, `--`) are not allowed.
- Casts, `sizeof`, function calls, `? :`, and the comma operator (`,`) are not allowed

Shortcuts

You may find the following shortcuts useful.

General Shortcuts

- Clicking on hex numbers selects them. You can copy such selections and edit some of them (such as data in Memory windows). The value of the current selection is shown in the Status window. Some menu commands that require numbers operate on the selection if there is one.
- If you type anywhere when the front window doesn't accept keystrokes, the keystrokes go to the Listener window.
- `⌘-L` selects the Listener window.

In the CPU Window

- Clicking on the PC label (“PC”) opens a code window or scrolls one to reveal the instruction at the PC.
- Clicking on the mode labels opens and closes that pane of the window.
- Clicking on the status flags toggles them.
- Clicking on the mode cycles through the mode.
- Clicking on a register name opens a window at that address.

In Code Windows

- Clicking on the target of a “B” or “BL” instruction opens a new code window or scrolls to show the target.
- ⌘-clicking on the breakpoint diamond opens the breakpoint dialog where conditional breakpoints can be set.
- ⌘-clicking in the PC column sets the PC to that instruction.
- Clicking in the PC column sets a one-time breakpoint at the instruction and goes. The one-time breakpoint lasts until it is reached or you rerun.
- Clicking on the opcode toggles the display of an instruction between “normal” and “DCD”. In the “DCD” form you can change the hex value of the instruction. If you change the value, the change persists until you choose the Rerun menu command.
- ⌘-clicking on the opcode no-ops the instruction.

In the NewtonScript Listener Window

- ⌘-3 through ⌘-9 type strings from the Commands menu (see “Command Keys for the Listener” on page 3-11) which you can override (see the next section).
- ⌘-left arrow moves the insertion point to the beginning of the current line, and ⌘-right arrow moves the insertion point to the end of the current line.
- Shift-arrow key extends the selection in the direction of the arrow key.

Customizing NewtonScript Listener Shortcuts

You can create a file called Listener Commands and put it in the Newtsbug directory in order to implement your own Listener shortcuts.

When Newtsbug is launched, it searches in its folder for this file. You can replace some or all of the default commands for ⌘-3 through ⌘-9 in the Memory menu. If the file is not found, default commands are used. Following define the format of the file:

Listener Commands is a text file.

Each line becomes a command in the Memory menu, with the first being cmd-3.

Using Newtsbug

If the last letter is '\', then the command will not be executed immediately, which allows you to type other text after the command.

In the Stack Trace Window

- Clicking on a routine reveals the instruction at the return address in that routine.
- ⌘-clicking on a routine sets a one-time breakpoint at the return address and goes.

Debugging Examples and Tips

Here are some examples of using Newtsbug.

Converting Values

I want to convert between hex and decimal

Select the constant and then choose the Convert menu item or just select the value and look in the Status window, which always displays the most recently selected hex value in both decimal and ASCII.

Examining Parameters

I want to check the parameters to a routine

The first four parameters are passed in R0, R1, R2, R3 and the result is returned in R0. Additional parameters are passed on the stack. Select the value in SP and press ⌘-M.

C++ non-static member function have `this` as an implied first parameter. The first parameter is passed in R0 unless the routine returns a struct larger than 4 bytes; in which case, a pointer to the return area is passed in R0. See also section 5 of the ARM Technical Specification manual.

Actually (for non-static function members) "this" will be in R0 unless the function returns a struct (or union of class). In that case R0 will be a pointer to the return struct and R1 will be "this."

I crashed in a routine and want to find out what parameters caused the problem.

- 1) Look at the beginning of a routine to see if the parameters (R0-R3) were saved in permanent registers.
- 2) If that doesn't help, set the PC to the return instruction (LDM) and step to get back to the previous routine. At this point, you can often rerun the call by changing the PC.

Using Newtsbug

Alternatively, you can set the PC to the new return instruction and back out another level.

Stack Frames

I want to look at a local in the stack frame.

- 1) Look for a place which assigns or reads the local for a procedure call and then use the procedure call to tell you what offset/register the local is at.
- 2) Look for a place at the beginning of the function where the local is initialized with a constant or a parameter.
- 3) Add a dummy procedure call to your function which takes the local as a parameter.

Compiler Idioms

I want to learn common idioms of the compiler so I can ignore them

Standard Entry saves registers used and sets up a stack frame.

```
MOV R12, SP
STMDB SP!, {R4-R7, R11-R12, LK-PC}
SUB R11, R12, 4
```

Standard Exit restores registers, fixes up the stack, and returns.

```
LDMDB R11, {R4-R7, R11, SP, PC}
```

Standard Exit is sometimes optimized into a tail call.

```
LDMDB R11, {R4-R7, R11, SP}
B LastProcedureCallInMethod
```

Small structures are copied using load multiple.

```
LDMIA R0, {R3, R12}
STMIA R2, {R3, R12}
```

Anytime you use a RefVar, a constructor will be inserted.

```
MOV R0, Rx
ADD R0, SP, #xx
BL __ct__6RefVarFC1
MOV Rx, R0
```

And a destructor will be inserted at the end of the block.

Using Newtsbug

```
ADD R0, SP, #xx
MOV  R1, #2
BL  __dt__6RefVarFv
```

(Note that passing a Ref to a function taking a RefArg will implicitly construct/destroy a temporary RefVar.)

Virtual method calls jump through an array of method pointers.

```
MOV LK, PC
LDR PC, [Rx, #xx]
```

Reading a short is done by reading a shifted long.

```
LDR R0, [SP, #xx]
MOV R0, R0, ASR 16
```

Writing a short is done a byte at a time (which is why you should avoid shorts).

```
STRB R0, [SP, #xx+1]
MOV R0, R0 ASR 8
STRB R0, [SP, #xx]
```

How to find out the Real PC

If the Newton stops by itself due to a breakpoint, a data abort, or some other reason, the current PC value is correct. However, when you manually stop the Newton from Newtsbug, the current PC is always inside `MainSCCInterrupt` handler. You may really want to know where the base level program (most likely user mode) that was interrupted is.

1. Find out the current CPU mode by looking at the CPSR field.
It will be either FIQ mode or IRQ mode, depending on whether you were using the built-in serial port or the serial port on a PCMCIA card.
2. Look at the SPSR field of the current CPU mode to see the mode when the break occurred.
3. Select the LK register of that CPU mode.
4. Press \mathfrak{H} -P to make Newtsbug open a code window showing the routine that contains the line referred to in the LK register.
The line referred to in the LK register is underlined. The line above it will probably be a BL instruction.
5. Click on the procedure name part of that BL instruction to make Newtsbug open a code window showing that procedure.
That is the procedure that contains the actual PC. (Note that Newtsbug cannot tell exactly where in that procedure the PC was.)

Debugging Layout and Placement

You can alter the Newton system temporarily so that all views show borders. To do so:

1. Display the Procedure Browser using the Procedure command from the Procedures menu.
2. Type `TView` in the left box and `Draw` in the right box.
3. Select the `Draw` method whose first parameter is `TBaseRegion`, and click OK.
A Procedure Browser showing that method's code displays.
4. Scroll to the bottom of that procedure.
About twelve lines from the end, you'll see a `TEQ` and a `BEQ`.
5. Change the `BEQ` to a no-op by \mathbb{H} -clicking on `BEQ`. (You can change it back to a `BEQ` by \mathbb{H} -clicking again on the `BNV`.)
6. In the Status window, click on `Go` to start the Newton system running again.

From now on every view will have a gray border.

Using Newtsbug

Index

Numerals

32-bit mode 1-1
680x0 Mac OS computers used with debugger 1-1

A

Add Time Stamps option 3-6
ARM Assembler 1-1

B

baud rate
 setting 3-8
Beep on Stops option 3-6
Breakpoint Conditions dialog 5-2 to 5-3
Breakpoints
 and stepping 4-2
breakpoints
 setting 5-1
breakpoints in ROM 2-3

C

cable needed to connect to Newton 2-2
choosing a serial port 3-8
Clear menu item 3-4
Close menu item 3-2
Close Port button in Status window 4-1
Code windows 4-6 to 4-8
 shortcuts 5-5
conditional breakpoints 5-1, 5-2 to 5-3
Config menu 3-21
Copy menu item 3-4
Copy Target Screen menu item 3-4
CPU menu item 3-20
CPU window 4-3
Cut menu item 3-4

D

Debugger Connection
 opening 2-4, 2-5
debugger traps 5-2
Default Stdio On configuration setting 3-21
disconnecting the Newton 2-5

E

Edit Menu 3-4
Enable Package Symbols configuration setting 3-22
Enable Stdout configuration setting 3-21
ending a debugging session 2-5
expression syntax 5-4

F

File menu 3-1
Find menu item 3-18
FPE Registers menu item 3-20
FPE Registers window 4-5
Frame menu item 3-15
Frozen Newton Reconnect menu item 3-3

G

Go button in Status window 4-2

H

Heap menu item 3-18
hex converter in Status window 4-3

I

image file
 “image” and “high” 2-1
 opening 3-1

Inspector 4-6

L

Listener window *see* NewtonScript Listener window
 Load Memory menu item 3-2
 Log Breaks option 3-6

M

Memory menu 3-14
 Memory menu item 3-14
 memory needed 1-1
 Modem Port or Printer Port option 3-8
 monitor size 1-2

N

Newton MessagePad 120 2-3
 NewtonScript Listener menu command 3-21
 NewtonScript Listener window 4-6
 customizing shortcuts 5-5
 shortcuts 5-5
 Newtsbug
 quitting 2-5
 Newtsbug Connection
 opening 2-1, 2-4, 2-5
 NTK 1-1
 NTK Inspector 4-6

O

Objects folder 2-6
 Open Image button in Status window 4-1
 Open menu item 3-2
 Open Port button in Status window 4-1
 operating system required 1-1

P

package
 alias to package file 2-3
 availability of symbols 2-4
 file name 2-5
 problems with finding symbols 2-6
 Paste menu item 3-4

PC

 finding the real value 5-8
 permanent breakpoints 5-1
 PowerPC computer used with debugger 1-1
 Preferences menu item 3-5
 printf
 preparing for use 2-3
 Project Settings menu item of NTK 2-3

Q

Quit button in Status window 4-2
 Quit menu item 3-4
 quitting Newtsbug 2-5

R

Register Names menu item 3-4
 Rerun button in Status window 4-2

S

Save As menu item 3-2
 Save Memory menu item 3-2
 Script Heap menu item 3-19
 Select All menu item 3-4
 serial PCMCIA card 2-2
 serial port
 choosing 3-8
 setting baud rate 3-8
 Smash Heap Tags menu item 3-19
 Sort by Address menu item 3-19
 Sort by Size menu item 3-19
 Sort by Task menu item 3-20
 stack frame
 examining locals 5-7
 Stack Trace From menu item 3-21
 Stack Trace menu item 3-20
 Stack Trace window 4-5
 shortcuts 5-6
 Status menu item 3-20
 Status window 4-1
 stdin/stdout/stderr menu item 3-21
 Stdin/Stdout/Stderr window 4-6
 Step button in Status window 4-2
 stepping
 and breakpoints 4-2
 in ROM 2-3
 Stop button in Status window 4-2

Stop on Aborts configuration setting 3-21
Stop on Debug traps option 3-6
.sym (symbol) file 2-5, 2-6
symbols
 loading 2-4
 problems with finding 2-6
system requirements 1-1

T

temporary breakpoint 5-1
temporary breakpoints 5-1

U

unconditional breakpoints 5-1
Undo menu item 3-4

W

Windows menu 3-20

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 Pro 630 printer. Final page negatives were output directly from the text and graphics files. Line art was created using Adobe[™] Illustrator. PostScript[™], the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type is Palatino[®] and display type is Helvetica[®]. Bullets are ITC Zapf Dingbats[®]. Some elements, such as program listings, are set in Apple Courier.

WRITER

Jonathan Simonoff

ILLUSTRATOR

Peggy Kunz

EDITOR

MP McKowen

PRODUCTION EDITOR

Gerry Kane

PROJECT MANAGER

Gerry Kane