Newton C++ Tool

Newtsbug User’s Guide

1.0

May 16, 1996 Jonathan Simonoff

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

Contents

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.

Newtsbug runs on Mac OS-based computers.

Since Newtsbug is a low-level debugger, it works on compiled code, and does not display higher-level language symbols. It does, however, display 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 . The Newtsbug Listener has a command, Command-2, which is equivalent to the NTK Inspector’s command that downloads a package.

System Requirements

Newtsbug has the following system requirements:

- For 680x0 Mac OS computers, you need a model that is at least as fast as a Macintosh IIci. Any PowerPC computer running the Mac OS will work.
- 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 that 14 inches or larger. It can be black and white, gray scale, or color.

Getting Started

You can use Newtsbug for debugging C code and NewtonScript code.

What You Need

You need the following to debug with Newtsbug:

- A Newton using a 2.0 (or later) system, or which has a stable flash ROM that includes the ROM code needed for Newtsbug. You should use a power adaptor rather than the Newton's battery, because the Newton will not sleep when stopped in the debugger.
- A pair of image files, a .image and a .high file that match the system on your Newton.
- A Mac OS-based computer.
- A serial cable.
- The package you want to debug loaded on the Newton.
- The package (.pkg) file or its alias in Newtsbug's folder. You almost always use an alias.
- The file with symbol information for the package in a place where Newtsbug can find it. See "Files Newtsbug Needs to Find" on page 5 for information.

You may want to have 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 the Newton; we recommend that developers use this. Other cards may work as well, but haven't been tested as of this writing.

Beginning Debugging

1. Load Newtsbug Connection on your Newton device. Newtsbug Connection is included on the CD that contains Newtsbug.

2. Create an alias for your package file and put it in the Newtsbug folder on your Mac OS-based computer. See “Files Newtsbug Needs to Find” on page 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.
3. Run Newtsbug.
4. When Newtsbug asks for an image file, click on the Cancel button.
5. Choose the Preferences command from the Edit menu.

   Newtsbug displays a dialog box that lets you set various features of Newtsbug.
6. 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.
7. 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 is slower than a Macintosh Quadra, choose something slower than 57600.
8. If you want to 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.) Each page is 4K of RAM. More memory allows more breakpoints. 8K is required by the system to support stepping in ROM. Each additional 4K will give at least one breakpoint. More breakpoints in the same 4K range don't use any additional memory. For example, 20K (8K for the system and 12K for breakpoints) will allow you to set breakpoints in three different 4K pages. You can reset the amount of memory available by using the Preferences command again.

Note

If you are using a Newton MessagePad 120 running Newton 2.0, 2.1, or 2.1D 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. ◆

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

Warning

If the image file you choose does not match the system that is on the Newton device, you are warned, but Newtsbug allows you to continue. However, doing so can cause undetermined problems, including Newtsbug errors that will force you to reboot your Mac OS-based computer. ▲

11. On the Newton, open the Newtsbug Connection application, which will be in the Extras drawer.
12. Select the same baud rate you chose in Newtsbug.

13. Tap on Connect.

You should see a Listener window on the Mac OS-based computer.

Making Sure Your Package Symbols are Available

When you start a debug session, Newtsbug scans the Newton device for currently loaded package and looks for symbol files that re 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 need to reinstall 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 (see step 2 on page 4) to match the name in the error message and reinstall the package.

Ending a Debugging Session

To terminate a debug session:

1. Open Newtsbug Connection again.
   Now the button should say Close.
2. Tap on the Close 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.

Newtsbug uses a naming scheme to find the package file. It uses information in the package file to find the .sym file.

Naming the Package File So Newtsbug Can Find It

Newtsbug looks for the package file, or an alias to the package file, in its directory. It derives the expected name from the Name entry of the Package Settings icon of the Project Settings menu item. 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.

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.
- 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
- Newtsbug appends .pkg to the name.

Handling the .sym File

When Packer creates the package, it creates 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 for the Mac OS 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 these tactics. Either of them may work; try whichever is easiest for you first.

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

You can tell if your package symbols are being read successfully by watching the Newtsbug stdio window while you make the connection with the Newton. Newtsbug indicates when it successfully reads a package file and symbol file. (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 you are interested in.

Features

This section lists all the features supported by Newtsbug.

- Execution control
    - Stop target
    - Set breakpoints in ROM, packages
    - Set breakpoints with expressions (e.g. R0== 0x00000000)
    - Step and step over in ROM or packages
    - Can set breakpoint while target is running (that is, you don't need to stop the image)
- Exceptions Handling
    - Stop on aborts
- Memory Related
    - See heaps
    - See memory
    - Modify memory
- Code Related
    - See code (ROM and packages)
    - Change instructions
- Stdio
    - Stdio window; all `printf` calls are output to this window
    - Log files
- Listener window
    - Cut, Copy, Paste, Clear, Select All, Undo
    - loadpackage (cmd-2)
    - Quick keys (cmd-2 through cmd-9), which can be customized
    - All NewtonScript `print` calls are output to this window
    - Time stamp for each line of text from target
    - Window contents are automatically saved in a file when session is closed
    - Log
- Choose any available serial channels
    - 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
- Start debugging after target crashed

- Miscellaneous
  - See/Modify register
  - Stack trace
  - Copy target screen
  - CPU window
  - FPE registers
  - Event tracing

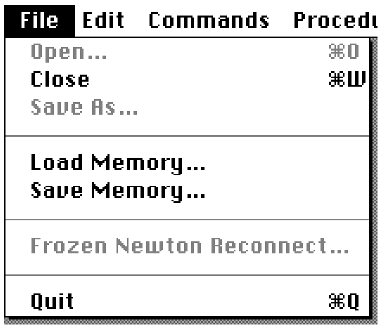
Newtsbug Menus

The sections in this chapter describe the Newtsbug menus.

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 3-1 The File Menu



Open (⌘-O)

The Open menu item lets you open, load, and run a new image file.

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 stdio windows such as the Listener, stdin/stdout/stderr window, heap windows, and Spy windows.

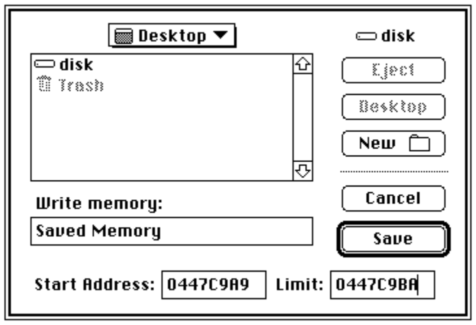
Load Memory...

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

Save Memory...

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

Figure 3-2 The Save Memory Command



Write memory: Saved Memory

Start Address: 0447C9A9 Limit: 0447C9BA

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 us usual.
2. Disconnect the serial cable (make sure the target 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. Take the Newton to anywhere and do anything as normal, except don't use the serial port that was used in step 1 for setting of 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 sit in the low level debug loop waiting for the Newtsbug to connect to it. The battery may be used up very quickly. ▲

5. When the Newton freezes, connect it Newton to Mac OS computer with a serial cable; launch Newtsbug, open a proper image, and use the Connect to Frozen Newton 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.
7. After finishing debugging, you should reset the Newton, because the Newton's internal state may be bad.

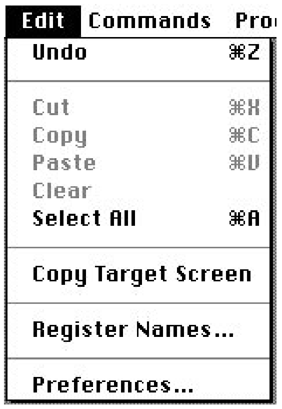
Quit(-Q)

The Quit menu item quits the program and ends this debugging session.

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 Newtsbug functions. The standard commands are not documented here.

Figure 3-3 Edit Menu



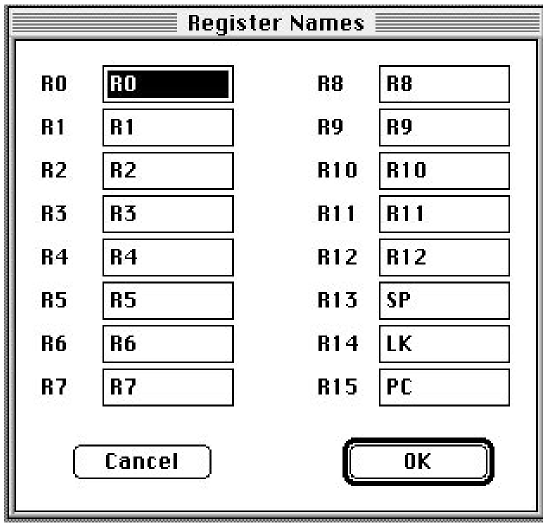
Copy Target Screen

The Copy Target Screen command copies the screen of the Newton to the clipboard.

Register Names...

The Register Names command puts up a dialog that lets you name the registers. The dialog is shown in Figure 3-4.

Figure 3-4 Register Names Dialog



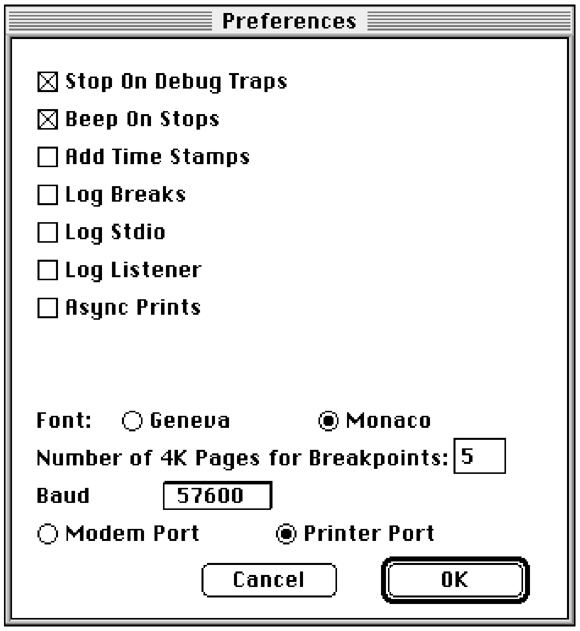
Register Names

R0	R0	R8	R8
R1	R1	R9	R9
R2	R2	R10	R10
R3	R3	R11	R11
R4	R4	R12	R12
R5	R5	R13	SP
R6	R6	R14	LK
R7	R7	R15	PC

Preferences...

The Preferences command puts up a dialog that lets you set various features of Newtsbug. Figure 3-5 shows the dialog. The options are described below the figure.

Figure 3-5 Preferences Dialog


Preferences

☒ Stop On Debug Traps
☒ Beep On Stops
☐ Add Time Stamps
☐ Log Breaks
☐ Log Stdio
☐ Log Listener
☐ Async Prints

Font: ○ Geneva ● Monaco
Number of 4K Pages for Breakpoints: 5
Baud 57600
○ Modem Port ● Printer Port

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 when the image is stopped are not included.

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 display a log line if a break point is hit with an expression condition met (see "Expressions" on page 38). It doesn't matter whether the "Break after" condition is met.

Newtsbug will display a line similar to this:

```
2/10 break at PC=   100530  TTracer::TTracer(char*) + 12
```

This may be useful to trace the program flow, especially if you turn on Add Time Stamp option, as well.

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 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 below). 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. 8K is required by the system to support stepping in ROM. Each additional 4K will give at least one breakpoint. More breakpoints in the same 4K range don't use any additional memory. For example, 20K (8K for system and 12K for breakpoints) memory will allow a user to set breakpoints in three different 4K pages.

Note
If you are using a Newton MessagePad 120 running Newton 2.0, 2.1, or 2.1D, 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, called 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 is slower than a Macintosh Quadra, choose something slower than 57600.

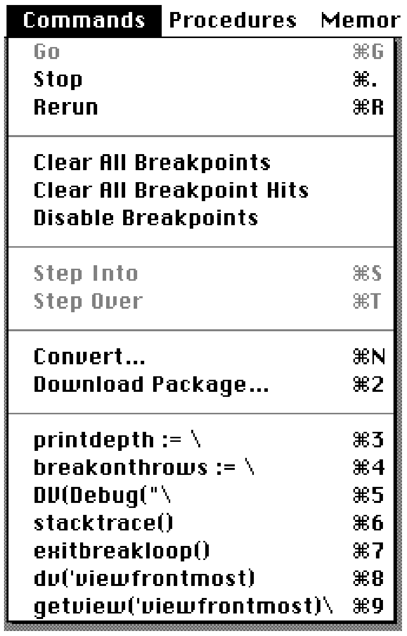
Modem Port or Printer Port

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 3-6 Commands Menu



Go (-G)

The Go command causes a suspended (Stopped) program 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). Interrupts can be blocked in the ARM CPU or by the control ASIC
- 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 using the reset button on the Newton. 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 37.)

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 37 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)

These commands are used to advance program execution one instruction, or step, at a time. The Step Into and Step Over commands 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 29). 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.

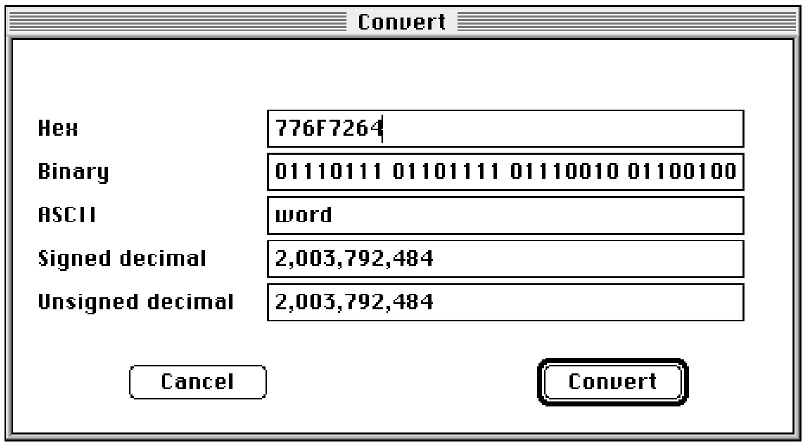
If the instruction at the current PC is not a BL instruction, the menu items show as Step (Into) and Step (Over).

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 reboot 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 3-7 Convert Dialog



Hex: 776F7264
Binary: 01110111 01101111 01110010 01100100
ASCII: word
Signed decimal: 2,003,792,484
Unsigned decimal: 2,003,792,484

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.

Command Keys for the Listener

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

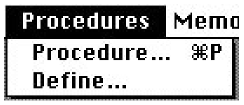
Newtsbug will type the characters in the command you select in the current line in the Listener window. If the last character in the string is not a back slash (\), Newtsbug deletes any text that might be on the current line, enters the command, and adds a return character. If the last character in the string is a back slash, Newtsbug does not add a return character.

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.

Procedures Menu

This menu is used to find or define procedures so that you can quickly access them using code windows. Aside from the menu commands 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 34 for information on those windows.

Figure 3-8 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.

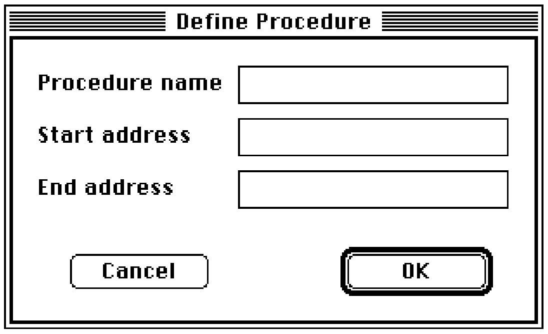
Newtsbug's search for a symbol is not case sensitive and a prefix of a symbol can be used (for example, "`asyncc`" to find `AsyncCallback`).

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

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.

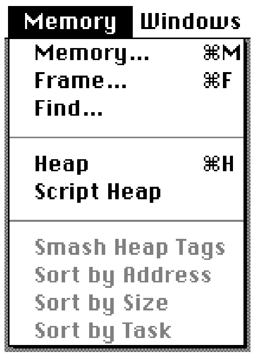
Figure 3-9 Define Procedures Dialog



Memory Menu

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

Figure 3-10 Memory Menu



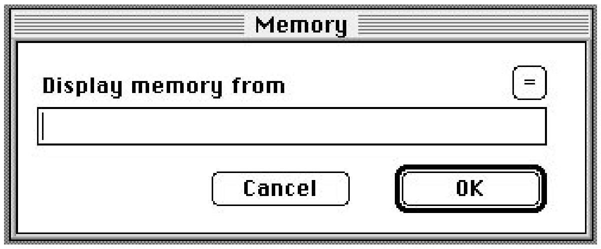
Memory...

Memory windows let you display 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 3-11 shows the dialog. You can enter:

- An address in hex
- A symbol such as a global variable name
- An expression; see “Expressions” on page 38 for details

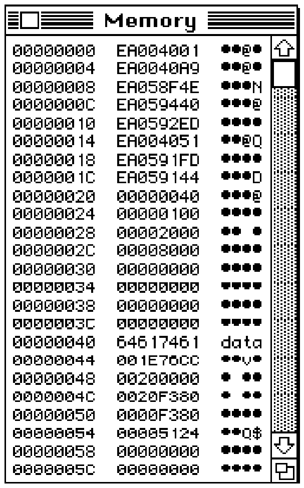
You can press the = button to evaluate the expression

Figure 3-11 Memory Dialog



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 ±4K.

Figure 3-12 Memory Window



```
00000000  EA004001  ●●@●
00000004  EA0040A9  ●●@●
00000008  EA058F4E  ●●●N
0000000C  EA059440  ●●●@
00000010  EA0592ED  ●●●●
00000014  EA004051  ●●@Q
00000018  EA0591FD  ●●●●
0000001C  EA059144  ●●●D
00000020  00000040  ●●●@
00000024  00000100  ●●●●
00000028  00002000  ●● ●
0000002C  00008000  ●●●●
00000030  00000000  ●●●●
00000034  00000000  ●●●●
00000038  00000000  ●●●●
0000003C  00000000  ●●●●
00000040  64617461  data
00000044  001E76CC  ●●v●
00000048  00200000  ● ●●
0000004C  0020F380  ● ●●
00000050  0000F380  ●●●●
00000054  00005124  ●●Q$
00000058  00000000  ●●●●
0000005C  00000000  ●●●●
```

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

The hex values displayed in the window can be edited. 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.

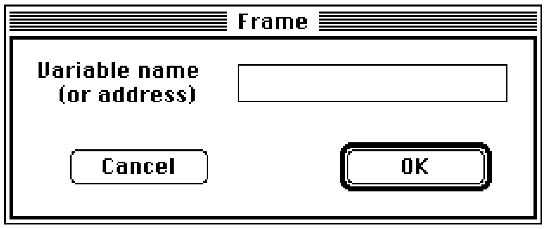
Table 3-1 The Frame Command

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		Opens a window displaying the frame or array
The address of a Ref of a frame or array	The number is treated as an address, and a memory window is opened at the selected address	
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	The number is treated as an address, and a memory window is opened at the selected address	

[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 command, Newtsbug displays the dialog box shown in Figure 3-13.

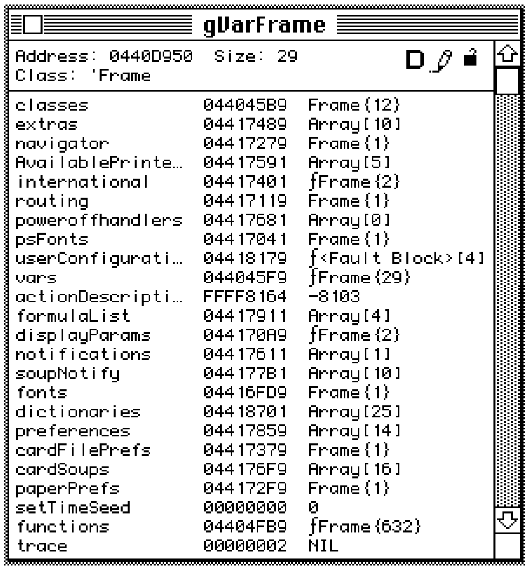
Figure 3-13 Frame Dialog



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 3-14 shows a frame window for `gVarFrame`.

Figure 3-14 Frame Window



```
Address: 0440D950  Size: 29
Class: 'Frame
classes              044045B9  Frame{12}
extras               04417489  Array[10]
navigator            04417279  Frame{1}
AvailablePrinte…     04417591  Array[5]
international        04417401  ƒFrame{2}
routing              04417119  Frame{1}
poweroffhandlers     04417681  Array[0]
psFonts              04417041  Frame{1}
userConfigurati…     04418179  ƒ<Fault Block>[4]
vars                 044045F9  ƒFrame{29}
actionDescripti…     FFFF8164  -8103
formulaList          04417911  Array[4]
displayParams        044170A9  ƒFrame{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  ƒFrame{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.

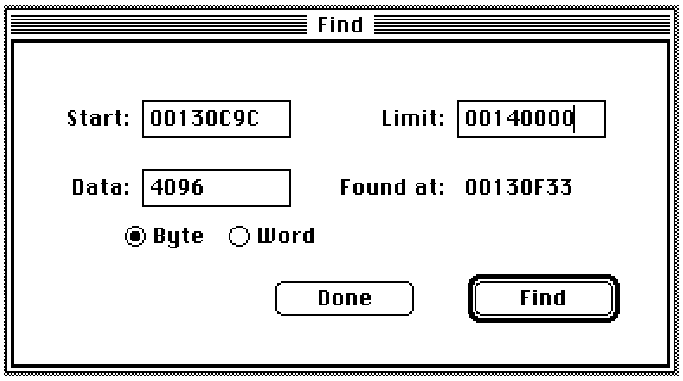
- Clicking on the Ref in the middle column selects it like any other Newtsbug number.
- Clicking on a tag in the left column or the value in the right columns is the same as selecting the Ref and invoking the Frame command with the shift key up; that is, it opens a Frame or memory window on the object 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 3-15 Find Dialog



Start: 00130C9C Limit: 00140000
Data: 4096 Found at: 00130F33
[✓ Byte] Word

You can click the Find button successively to continue searching. To start the search over, you need to close and re-open the window, or change start, limit, or data.

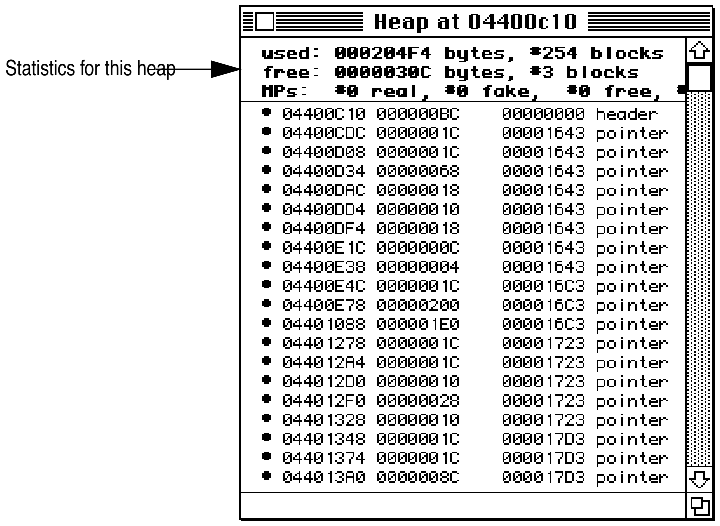
Heap

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 target 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 target go. ◆

Figure 3-16 Heap Window



```
used:  000204F4 bytes, *254 blocks
free:  0000030C bytes, *3 blocks
MPs:   *0 real, *0 fake,  *0 free, *
• 04400C10 000000BC    00000000 header
• 04400CDC 0000001C    00001643 pointer
• 04400D08 0000001C    00001643 pointer
• 04400D34 00000068    00001643 pointer
• 04400DAC 00000018    00001643 pointer
• 04400DD4 00000010    00001643 pointer
• 04400DF4 00000018    00001643 pointer
• 04400E1C 0000000C    00001643 pointer
• 04400E38 00000004    00001643 pointer
• 04400E4C 0000001C    000016C3 pointer
• 04400E78 00000200    000016C3 pointer
• 04401088 000001E0    000016C3 pointer
• 04401278 0000001C    00001723 pointer
• 044012A4 0000001C    00001723 pointer
• 044012D0 00000010    00001723 pointer
• 044012F0 00000028    00001723 pointer
• 04401328 00000010    00001723 pointer
• 04401348 0000001C    000017D3 pointer
• 04401374 0000001C    000017D3 pointer
• 044013A0 0000008C    000017D3 pointer
```

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

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.

Sort by Address

Sorts the items in the selected heap window by address.

Sort by Size

Sorts the items in the selected heap window by size.

Sort by Task

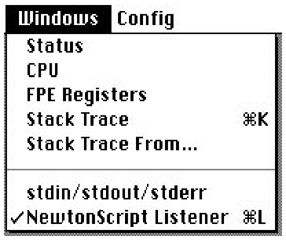
Sorts the items in the selected heap window by task.

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.)

The bottom section of the menu, below the line, can have different contents depending on what windows are displayed. These windows are described in Chapter 4, “Windows” on page 29.

Figure 3-17 Windows Menu



Status

This menu item brings the Status window to the front. See “Status Window” on page 29 for more information.

CPU

This menu item brings the CPU window to the front. The CPU window shows the state of the CPU. See “The CPU Window” on page 31 for more information.

The FPE Registers

This menu item brings the FPE Registers window 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 “The Stack Trace Window” on page 33 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.

NewtonScript Listener

This menu item brings the Listener window to the front.

Config Menu

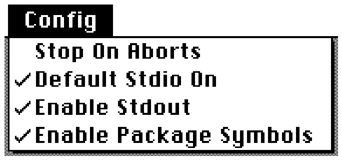
The Config menu lets you use Newtsbug to configure images in various ways.

IMPORTANT
Newtsbug may have trouble running with certain Config menu settings.
In particular:

- Stop on Aborts should be off
- Default Stdio On should be on
- Enable Package Symbols should be on unless you aren't debugging a package

In general, you should not change these settings unless you have a particular reason to do so. ▲

Figure 3-18 Config Menu



Stop on Aborts

Useful for finding bugs.

Default Stdio On

Standard I/O interferes with interrupts; set this option off to prevent standard I/O.

When this item is checked and you are running a debug image, a NewtonScript Listener window opens. This is similar to the Listener you can use with NTK.

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

Enable Stdout

Leave this option off if you only need stdio for the Listener.

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 Newtsbug 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 Newtsbug preferences file.)

Windows

This chapter describes the Newtsbug windows.

Status Window

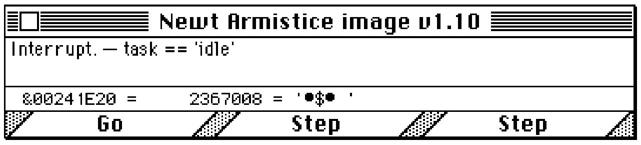
The bottom of the Newtsbug Status window shows one, two, or three buttons depending on the state of your program. When the Status window is first opened (when the image is launched) the window displays just one button at the bottom of the window, which can change depending on the state of the program:

- If you haven't yet selected an image, the button says Choose 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, 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 connected to a Newton, then the image is running, and the button says Stop. If you click on it, it stops the image and the Newton, and you enter the debugger.

Go and Step Buttons

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

Figure 4-1 Status Window



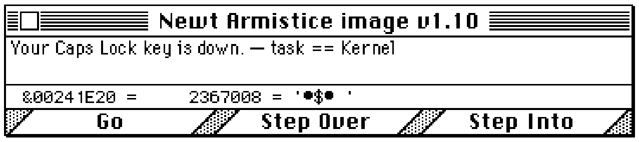
```
Newt Armistice image v1.10
Interrupt. — task == 'idle'

&00241E20 =      2367008 = '•$• '
```

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 Newtsbug will either lose the connection with the Newton or crash. ▲

Figure 4-2 Status Window Showing Step Over



```
Newt Armistice image v1.10
Your Caps Lock key is down. — task == Kernel

&00241E20 =      2367008 = '•$• '
```

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

Handy Hex Converter

In addition to indicating the execution state of the program (Running...) or the cause of program suspension, the Status window displays the value that you have most recently selected in one of Newtsbug's windows. This area of the Status window also functions as a hex converter: the hex last selected value field can be edited and the equivalent decimal and ASCII values will be displayed. The hex value can also be selected and copied.

Stdin/Stdout/Stderr Window

This window shows error messages, `printf`s, 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 10). You can also set up Newtsbug so that it logs information to a file (see “Log Stdio” on page 14).

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.)

The NewtonScript Listener Window

This window is very similar to the NTK Inspector. You can type in NewtonScript and when you press the Enter key the current line is transmitted to the Newton device for execution. You can also select one or more lines and press Enter.

In addition:

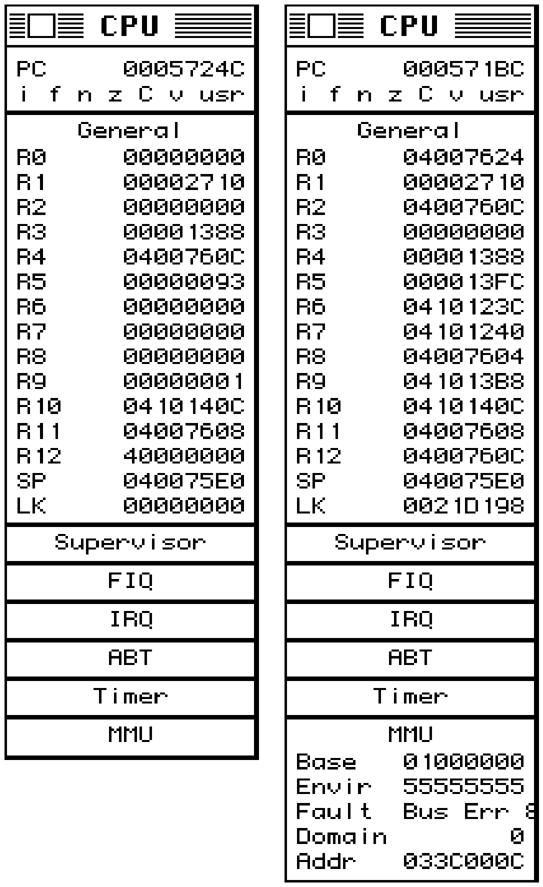
- You can use the Edit menu commands.
- You can save the contents to a file (see “Save As...” on page 10)
- You can log the contents to a file (see “Log Listener” on page 14)
- You can use items from the Commands menu to insert strings (see “Command Keys for the Listener” on page 17)
- You can have output from the Newton time-stamped (“Add Time Stamps” on page 13)
- There are a number of shortcuts (see “In the Listener Window” on page 40)

The 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 accidently select fields in the panes, and if you select certain fields and then execute code, the results are unpredictable.

Figure 4-3 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
                         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. Command-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. ◆

The FPE Registers Window

This menu item displays a window showing the contents of the floating-point registers. They cannot be edited.

The 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 37.) 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.)

Code Windows

The Procedures 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.

Each procedure in your program, as defined by the image file, can be displayed as a separate code window. These windows are typically 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 Newtsbug 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 four ways:

- Click on a procedure name that is the operand of a Branch instruction.
- Select the procedure's name from the Procedures menu. This menu has a list of all open code windows.
- Select Procedure from the Procedures menu and type in the name or address (in hex). Searching for a symbol name is not case sensitive and a prefix of a symbol (name) can be used (for example, `asyncc` to find `AsyncCallback`).
- Use the Procedure Browser to find a procedure. See "Procedure Browser" on page 36 for information.

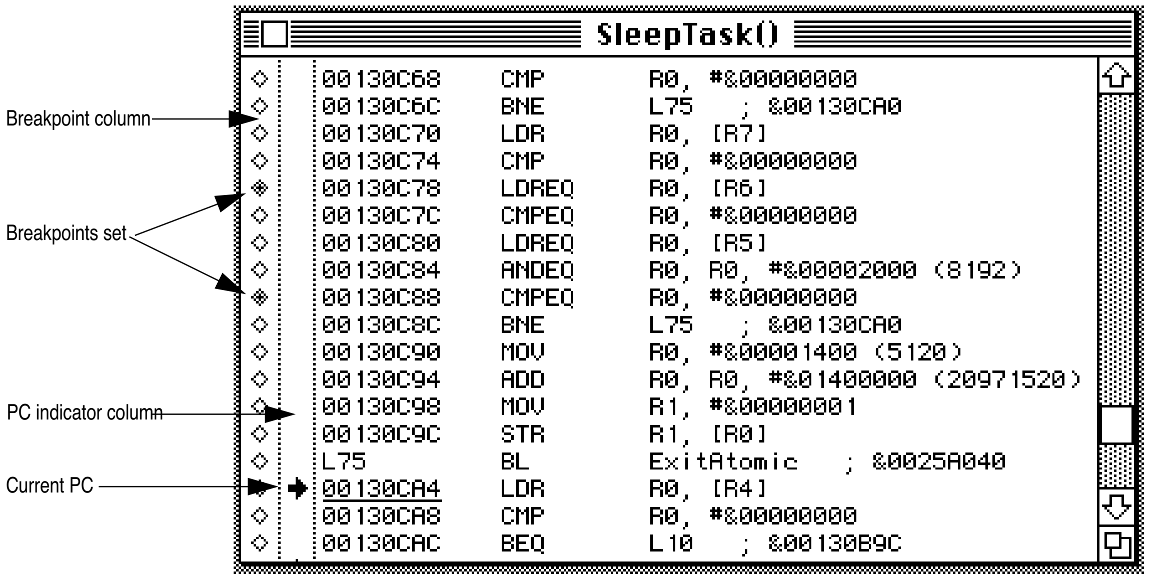
Code Window Contents

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.

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

Figure 4-4 Code Window Example



```
◇   00130C68    CMP      R0, #&00000000
◇   00130C6C    BNE      L75    ; &00130CA0
◇   00130C70    LDR      R0, [R7]
◇   00130C74    CMP      R0, #&00000000
◈   00130C78    LDREQ    R0, [R6]
◇   00130C7C    CMPEQ    R0, #&00000000
◇   00130C80    LDREQ    R0, [R5]
◇   00130C84    ANDEQ    R0, R0, #&00002000 (8192)
◈   00130C88    CMPEQ    R0, #&00000000
◇   00130C8C    BNE      L75    ; &00130CA0
◇   00130C90    MOV      R0, #&00001400 (5120)
◇   00130C94    ADD      R0, R0, #&01400000 (20971520)
◇   00130C98    MOV      R1, #&00000001
◇   00130C9C    STR      R1, [R0]
◇   L75         BL       ExitAtomic    ; &0025A040
  ➡ 00130CA4    LDR      R0, [R4]
◇   00130CA8    CMP      R0, #&00000000
◇   00130CAC    BEQ      L10    ; &00130B9C
```

Command-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 37.

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 command-click on the opcode to change the instruction to NOP.

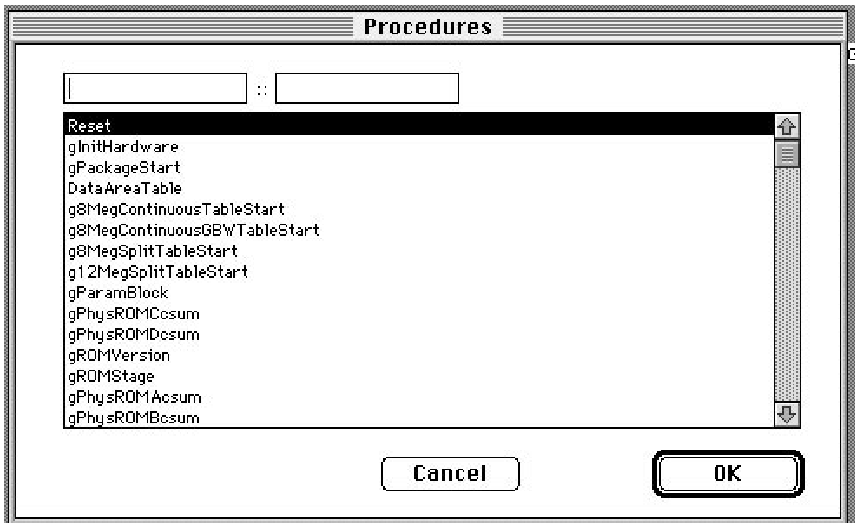
Unreachable instructions in C code are automatically shown in hex.

Procedure Browser

You can display the Procedure Browser by typing command-p. This browser lets you see all the current procedure names or look at the member functions of a given class.

Figure 4-5 shows the Procedure Browser.

Figure 4-5 Procedure Browser



```
Reset
gInitHardware
gPackageStart
DataAreaTable
g8MegContinuousTableStart
g8MegContinuousGBWTableStart
g8MegSplitTableStart
g12MegSplitTableStart
gParamBlock
gPhysROMCcsum
gPhysROMDcsum
gROMVersion
gROMStage
gPhysROMAcsum
gPhysROMBcsum
```

Clicking on a function causes Newtsbug to open a code window for that function.

Using Newtsbug

This chapter has information on how to use Newtsbug.

Breakpoints

The simplest way to set a breakpoint is to click on a PC indicator column in a code window. (See "Code Windows" on page 34.) A temporary breakpoint is set at the specified instruction and a Go command is issued. When the temporary breakpoint is hit, it is automatically cleared.

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. ◆

A permanent breakpoint is set by clicking in the diamond to the left of an opcode. Clicking on the diamond a second time clears the breakpoint.

You can see the breakpoint dialog by command-clicking a breakpoint. It contains:

- A Stop After check box.
- A place to enter a value. This can be used to stop only after the breakpoint has been hit the number of times indicated by the value. Alternatively, it can be turned off so that the breakpoint doesn't stop and just counts the number of times it's been hit.

Warning
Do not set a breakpoint in the MainSCCInterrupt handler because Newtsbug will either lose the connection with the Newton or crash. ▲

Expressions

You can use expressions in a number of places in Newtsbug. 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'`, `'sl'`, `'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 int`s.
- 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 `typedef`s 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

Viewing Frames that Refer to Packages

Two of Newtsbug's menu commands, the Frame Functions command in the Commands menu and the Frames command in the Memory menu print information about NewtonScript frames. Those frames may contain references to packages. If so, you can make the package information available to Newtsbug this way:

1. Make aliases for the packages you are interested in and place them in the Newtsbug folder. You must have a copy of the package file on your Mac OS-based computer.
2. Turn on Enable Package Symbols in the Config menu. (Keep this turned off unless you want to debug a package, because leaving it turned on will result in some objects in frame windows being displayed incorrectly.)

If you have trouble, here are two steps you can take:

- When you open your application package, check the Stdio window for information about what alias name is expected.
- Check if Newtsbug complains that the file content doesn't match.

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.
- Command-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.
- Command-clicking on the mode toggles 26/32 bit.

In Code Windows

- Clicking on the target of a "B" or "BL" instruction opens a new code window or scrolls to show the target.
- Command-clicking in the PC column sets the PC to that instruction.
- Clicking the opcode of an instruction sets a one-time breakpoint at the instruction and goes. The one-time breakpoint lasts until it is reached or you rerun.
- Clicking in 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.

In the Listener Window

- Command-3 through Command-9 type strings from the Commands menu, which you can override (see "Command Keys for the Listener" on page 17)
- Up-arrow and down-arrow cycles through the last fifteen executed commands, and re-submits them
- Control-u erases the contents of the current command line
- Command-left arrow moves the insertion point to the beginning of the current line, and Command-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 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 command-3 through command-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.

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.
- Command-clicking on a routine sets a one-time breakpoint at the return address and goes.

Debugging 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 where it 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 Command-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. 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__6RefVarFCl
MOV Rx, R0
```

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

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

(Note that passing a Ref to a function taking a RefArg will implicitly construct/destruct 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

When you stop the Newton from Newtsbug, the current PC is always inside `MainSCCInterruptHandler`. You may really want to know where the base level program (most likely user mode) that is interrupted is.

There is an easy way to find out this. First you need to find out what CPU mode was interrupted. Most likely it was user mode.

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. The SPSR field of the current CPU mode shows the mode when the break occurred.
3. The LK register of the CPU mode that was interrupted contains the PC of interest.
4. Select the LK register and do a command-P to make Newtsbug open a code window showing the routine of interest.