TABLE OF CONTENTS

Viewing Touched Nodes

Introduction

MallocDebug is a utility for measuring the dynamic memory usage of applications and for finding memory leaks.   You can use **MallocDebug** to measure and analyze all allocated memory in an application or to measure the memory allocated since a given point in time.   **MallocDebug** also contains a conservative garbage detector that can be used to detect memory leaks.

Preparing your Application

To run your application so that **MallocDebug** can measure its malloc usage, Command-Drag and release the application icon over the **MallocDebug** app icon. **MallocDebug** will start the application, causing it to dynamically link against a special version of malloc() that enables debugging. **MallocDebug** will only work with OPENSTEP applications.

To prepare your application for tracking which nodes are touched by various operations, use the **mdbsetup** command line program (see "Viewing Touched Nodes" below).

Using MallocDebug

To use **MallocDebug**, you must first select an application to monitor. The

Open menu item in the *Application* menu brings up the *Select* panel.   Only currently running applications which you own and which have been configured for use with **MallocDebug** will appear in the panel.   Select an application by double-clicking its icon, and **MallocDebug**'s application window will appear.

When the *All* option is selected in the main window, **MallocDebug** displays a list of all currently allocated nodes in your application.   These nodes have been allocated by one of the standard C allocation functions (malloc, realloc, calloc, or valloc) or one of NeXT's zone allocation functions (NXZoneMalloc, NXZoneRealloc, NXZoneCalloc).

Analyzing Malloc Usage

The browser in the main window displays the stacks that allocate memory in a hierarchical fashion, showing the uppermost procedure at the root of the browser, and methods and functions as children of their callers. Each element of the browser shows the amount of memory that has been allocated under that method or function, and you can switch between showing sizes in bytes and number of mallocs by clicking the *Sizes* check box. By decending the hierarchy in the browser, you can quickly determine where in your application the malloc usage is excessive. However, since your application may be quite large and be comprised of many subsystem, there are some further analysis tools provided in the *Analyze* submenu to help understand the malloc usage.

There is an option to invert the hierarchy by choosing the *Invert* command. This causes the leaves of the hierarchy to displayed in the first column of the browser, and as you click on a method , its parents are displayed in the column to the right. With the hierarchy inverted, you can find which methods and functions are allocating the bulk of your malloced memory.

There may be some methods and functions that occur in many places in the hierarchy, and you might want to know what is the sum of all the memory allocated by a particular method. You can collapse all the instances of a method into one browser item by use the *Make Flat* command. With this command, every method and function is displayed in a single list, sorted by malloc usage.   To undo the *Make Flat* command is the *Make Tree* command.

Sometimes when you are looking at the usage of a particular part of your application, there are mallocs that happen on behalf of a library that you use, or by the Objective-C runtime, in which you have no control over the memory usage of these subsystem that you use.   To ignore the memory usage under these subsystems, there are the *Forget Selection*, *Forget Path*, and *Forget*

Zone commands that remove certain stacks from the browser. The *Forget Selection* will remove all stacks that contain the method or function in the current selection of the browser. Thus if you want to remove everything allocated underneath the Objective-C caching mechanism, you could selected *__cache_fill* and then choose *Forget Selection*. The *Forget Path* is like the *Forget Selection* command except it doesn't forget every instance of the method in the hierarchy, only the ones in the selected path. The *Forget Zone* command will forget all mallocs allocated from a particular zone. To choose the zone, use the Zone Inspector panel.

To get back the information forgotten by the various Forget commands is the *Remember* command, which brings back all the stacks from the last time you execute the *Show* command.

There is an operation, which is almost the inverse of the forgetting operations, the Filter operation. Which you choose the *Filter...* command, the Find panel comes up, and you can type in some text to filter with and click the Filter button. What this does it to find all the stacks that contain the string from the Find panel, and only include those stacks in the output. Furthermore, the hierarchies are rooted the methods and functions that match your input string, so that you don't have to search all over the hierarchy to find the method you just filtered with. To undo the effects of this command, use the *Remember* command.

Performance Analysis

When you're in the performance phase of your development, and you come to the realization that you need to reduce your memory usage, one of the questions you ask yourself is: "For a given operation, what is the memory usage by the various parts of my application"?   For example, when launching your application, it would be nice if you could generate the following analysis:

Launching Draw in version 1.5

DrawDocument	140K
DrawApp	20K
Objective-C	40K
AppKit	30K
Defaults	8K
Total	228K

With this analysis, you can discover where the memory problems lie, and attack the largest offenders of excess memory usage.   Furthermore, as you make progress on your memory diet, you want to regenerate the analysis so that you can (hopefully) chart the progress over time.

MallocDebug provides a way to categorize the mallocs in your application. The Mapper panel and the Mapper submenu provides the user interface for both specifying which mallocs are assigned to which category.   In the example above, the categories would be DrawDocument, DrawApp, Objective-C, AppKit, and Defaults.

The Mapper panel is an editor for a document that specifies which mallocs are to be assigned to which categories.

Creating a new mapping

There are 3 ways to map a set of mallocs to a category: by zone, by path, or by text.   To map mallocs by zone, you specify in the Mapper panel   the zone option in the radio button , enter the zone name in the *Arg:* field (or choose a zone in the Zone Inspector panel, and then choose the *Enter Zone* command), enter the name of the category in the *Category:* field, then click the *Add* button.   This moves all the outstanding mallocs that came from the specified zone into the specified category.   To map mallocs by path, you select a path in the main window browser, and click *Enter Path* in the Mapper submenu, provide a category name, and click the *Add* button. This moves all the mallocs under the currently selected subtree in the browser to the specified category.   To map mallocs by text, you can either type a string into the *Arg:* field, or use the *Enter Text* command to enter the currently selected method in the browser, specify a category name, and then click the *Add* button. This moves all the mallocs that have this specified text as part of its call stack into the specified category.

As you add each mapping, a new line appears in the bottom browser of the Mapper panel, which the type of category, the category name, and the argument, which is the zone, path, or text depending on the type of category. In this browser, items can be cut, copied, or pasted via the standard *Cut*, *Copy*, and *Paste* commands in the Edit menu.   The order of the mapping is very importance, since each of the mapping commands are executed in order.

After you've specified your mappings, you click the *Apply* button to apply the mappings to the current malloc information.   In the upper browser of the Mapper panel, the categories are displayed. Clicking on an item in the Categories browser will cause the main window to display the mallocs that belong this the clicked-on category. To undo this operation in the main window, choose the *Remember* command from the Analyze menu.

You can save the mappings with the *Save* or *Save As* commands in the Mapper submenu into ".mapping" documents.   These document can be opened via the *Open...* command in the same submenu.

Finally, a report can be generated via the *Report...* command, which provides tab separated tables that can be imported into a spreadsheet or charting application.

Examining Zone Usage

The Zone Inspector panel (under the Tools submenu) provides either the size in bytes of the number of malloc nodes for each zone in your application.

Damaged Nodes

MallocDebug also detects nodes that have been written to incorrectly. If your application has written past the end of a node, a right arrow (`>') appears by the node. Similarly, if your application has written before the start of a node, a left arrow (`<') appears by the node. Many of these errors are the result of using the result of **strlen(s)** as the argument to malloc for a string instead of **strlen (s) + 1**. Damaged nodes are listed first in all sorting modes.

Finding Memory Leaks

To detect memory leaks, **MallocDebug** contains a *conservative garbage*

detector.
When the *Leaks* button is pressed, **MallocDebug** searches through your program's memory for pointers to each node.   Any node that cannot be referenced is displayed as a memory leak.   Since the garbage detector cannot know which words in memory are pointers, it is possible that an integer has the same value as a pointer to a given node.   In this case that node doesn't show up as a leak, even though it really is.   This is why the garbage detector is called *conservative*.   In practice, this problem is very rare.

The second caveat is that the garbage detector only searches for references to the *beginning* of each node.   If your program doesn't retain a pointer to the start of a node, but instead retains a pointer into the middle of it, that node will show up as a leak even though it really isn't one.

Measuring Memory Usage

MallocDebug can show you the memory usage of a given portion of your program. To begin measuring, press the *Mark* button. After exercising a portion of your program, press the *New* button to see the nodes allocated since the mark. Note that **MallocDebug** always shows you the nodes that are still *currently* allocated, so you will see only those nodes allocated since the mark that haven't been freed.

Viewing Touched Nodes

MallocDebug can show you which nodes are accessed (read or written) by your application. Knowing which nodes are touched by your application is most useful for tuning the use of different allocation zones, thus improving your program's data locality and minimizing its working set.   To learn more about using zones, look in /NextLibrary/Documentation/NextDev/Concepts/Performance.

To record which nodes are touched, **MallocDebug** must place each allocated node from the relevant zones on its own virtual memory page.   Because of this additional memory requirement, you have control over which zones have this per-node monitoring enabled.   After you link your application with **libMallocDebug.a** (see above), you must run the **mdbsetup** program on your application to enable per-node monitoring for various zones.   The command

```
        mdbsetup MyApp.app/Myapp -protectable <zone list>
```

enables the viewing of touched nodes within the zones listed in <zone list>.

The strings "ALL" or "NONE" may also be specified instead of a specific list of zones.

The command

```
        mdbsetup MyApp.app/Myapp -unprotectable <zone list>
```

enables the viewing of touched nodes within all zones *except* those listed in <zone list>.   For examples, specifying "`-unprotectable default ObjC`" will allow you to see nodes touched in all zones besides the default and Objective-C zones.

The command

```
        mdbsetup MyApp.app/Myapp -print
```

shows what zones within the application are enabled for touched node viewing.

After applying **mdbsetup** to your application, run the application and select it in **MallocDebug** as described above. To learn what nodes are touched for a given operation of your application, first press the *Protect* button. Then perform the operation in your application. While you are using the application, **MallocDebug** records which nodes are touched. To see this list, press the *Touched* button. To see what nodes have not been touched, press the *Untouched* button. To stop the recording of touched nodes, press the *Unprotect* button. Pressing *Protect* again cleans the slate of recorded nodes.

When touched nodes are being displayed, some new types of nodes are listed. Nodes marked with a '+' were allocated since the *Protect* button was pressed. Nodes marked with a '-' were allocated and freed since the *Protect* button was pressed.