# 3.3 Release Notes: Malloc Debug

This file contains release notes for the 3.2, 3.1, and 3.0 releases of Malloc Debug.   Items specific to the 3.1 release are listed first, and the Release 3.0 notes follow.   There are no items specific to the 3.3 or 3.2 releases.

## Notes Specific to Release 3.1

### New Features

The following new features have been added to MallocDebug since Release 3.0.

· **Touched nodes information.**
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**, 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 ªALLº or ªNONEº 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 ª`-unprotectable default ObjC`º 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.

## 3.0 Release Notes:

# Malloc Debug

MallocDebug, a new tool included in Release 2.0, is designed to help you understand and improve the dynamic memory usage of the applications you write.   MallocDebug consists of two components:

· a library containing a version of **malloc** that gathers statistics on memory use

· an application for examining those statistics

The MallocDebug application is located in **/NextDeveloper/Apps**.   Start the application and read the on-line help, which includes information on how to prepare your application for use with MallocDebug.

## Differences from standard malloc

The debugging version of **malloc** used by **MallocDebug** is implemented differently from the standard system **malloc** so that its internal data structures are much less susceptible to being damaged by aberrant programs.   This causes several incompatibilities with the standard **malloc**:

· The debugging **malloc** implements zones by tagging each node with a zone tag rather than actually allocating the node in a different region of memory. Consequently, locality of reference of a program which uses zones cannot be determined by examining the addresses of nodes in **MallocDebug**.

· The debugging **malloc** does not store information about nodes in the nodes themselves, but rather in an auxiliary data structure which is kept in a remote area of memory. This data is accessed by hashing the address of the node, but this information *cannot* be accessed from an address which is within the node. For this reason, the **NXZoneFromPtr** function will work correctly only when passed a pointer to the start of a node. Otherwise it will return **NX_NOZONE**. For the same reason the leak detector may report a node as a leak even if there is a pointer to some internal part of that node.