Why use write-through? Doesn't that degrade the performance, defeating the purpose of a RAM disk?

Writing through does cause a performance hit when writing to a RAM disk, but it maintains a safe RAM disk image file by ensuring that changes are written to a real disk file. Furthermore, reading from the RAM disk is where the greatest benefit occurs, since reading only looks at the RAM disk and has no need to access the RAM disk image file. A typical real-world example is compiling and linking an application. A compiler has to read all the files it compiles, some repeatedly, and writes a relatively small amount. By enabling the write-through feature of ramBunctious, you gain the advantage of fast reads from a RAM disk while maintaining the security of having new information in a real file on a real hard drive in the real world.

Can ramBunctious be used to feed the hungry, heal the sick, stop wars, and guarantee equal justice for all?

Yes. But the explanation is complex; if we gave a full explanation, the only people who would understand the answer would be you and us.

What are some typical uses for RAM disks?

Since we're software engineers, our typical uses for ramBunctious are heavily development-oriented. We typically have auto-mount enabled because we work on the same projects for days, weeks, or months at a time. We use disk-based RAM disks with write-through enabled to ensure that if our machines crash our projects are still safe. Bob occasionally customizes a RAM disk that is RAM-only for temporary files such as object files needed only until the project is linked, or for downloading programs from the internet to see if he likes them. All of Bob's RAM disks have the status windows open, but shrunk so only the disk activity icons are visible. Bob likes the new "Startup Items" feature. He has one RAM disk (write-through) that has source code; the startup items folder contains an alias to a second RAM disk image file. The second RAM disk (save-on-quit) has the project file on it, and an alias to the project file in the startup items folder. So when the source code RAM disk is started, the project RAM disk is automatically mounted and the system is ready for development. Elden typically has write-through enabled, and the status windows are kept closed; since he has hundreds of files, he likes the low file storage overhead provided by a RAM disk.

What was changed in ramBunctious version 1.1.2?

ramBunctious 1.1.2 was a bug fix release. System 7.6.1 introduced a conflict with huge FSWrite's on some configurations. ramBunctious now implements huge FSWrite's as a series of smaller FSWrite's to avoid the problem.

There was a tiny tweak made to Balloon Help.

What was changed in ramBunctious version 1.1.1?

ramBunctious 1.1.1 is primarily a maintenance release; a few bugs were fixed, and some incompatibilities were identified.

A conflict with Directory Assistant was identified -- see the Known Incompatibilities section.

Now the delete keys work in the "Enter RAM disk size" dialog.

The single feature that was added was a "Put Away" menu item in the File menu. This allows you to put away the RAM disk whose window is frontmost. You can still put away individual RAM disks via the RAM Disk menu, but now you can accomplish more with keyboard shortcuts, and you don't have to navigate through hierarchical menus.

What features were added in ramBunctious version 1.1?

The ramBunctious RAM disk driver was further optimized; it is now up to 30% faster than Apple's built-in RAM disk control panel.

ramBunctious now looks for a "Startup Items" folder on a newly-mounted RAM disk. It then tells Finder to open all the items in that folder. This can be a very convenient feature. For example, if you use a 2.2 MB RAM disk as a Netscape cache, you could include an alias to Netscape in the startup items folder. Then, when you're ready to hit the Web, you could start the RAM disk and Netscape would be launched automatically. If you want to temporarily disable this feature, hold down the "Shift" key while mounting the RAM disk.

ramBunctious is now a fat application; it runs natively on PowerPC Macs as well as 68K Macs.

A major focus for this update was to enhance ramBunctious's robustness. This involved several architectural modifications; these aren't visible, but the underlying code is more stable and will be easier to modify for future maintenance. For example, there are no more hard-coded string messages. All strings are now in 'STR#' resources, which will greatly simplify localization. Another example is a new ExitToShell() patch to guarantee that even if ramBunctious is forcibly exited, there will be no data integrity problem.

ramBunctious now uses Infinity Windoids for the settings windows. Thanks, Troy Gaul, for great-looking, easy-to-implement windoids.

There are a few minor user interface enhancements in this version. For example, there is a "Save Now" button in the settings window, and a "Save" menu item. The process for creating a new RAM disk has been streamlined as well; there's one dialog involved, and it includes a popup menu with common default RAM disk sizes. The write-through icon in the settings window now also indicates when the volume is "dirty". This dirty indicator can be a useful piece of information, especially if you want to perform a write-through manually on a save-on-quit RAM disk.

When a RAM disk is being initialized, a window now indicates the status. This is useful because, especially with large RAM disks, the startup process can take several seconds, and it's comforting to know precisely what the computer is doing during that time.