Bitwise Operator

by Matt Slot

This month's column outlines more fundamentals of Macintosh programming, including application frameworks and the components of a Macintosh application. Like the last one, this article doesn't contain any sample code to help you roll up your sleeves; rather, its lays a foundation of information that will continue to be relevant as you continue to learn.

Application Frameworks

To simplify the task of prototyping and finishing an application, many programmers prefer to use "frameworks," which handle most of the mundane tasks of implementing software. Without a framework, the task of creating a document window may take 10 to 20 instructions; a framework provides interfaces to do this with just 1 to 3 instructions. In general most frameworks are written in C++, which lends itself well to layered software.

There are a couple of popular frameworks, and several smaller ones, that a programmer can choose from. Symantec C++ comes with the THINK Class Library or TCL, Metrowerks CodeWarrior bundles PowerPlant, and Apple provides MacApp. TCL applications tend to be rather large due to the integrated nature of the framework. PowerPlant is a much newer design (incidentally by the creator of TCL), and provides marginally better segmenting for smaller applications.

MacApp is a commercial tool from Apple which provides large amounts of source code and development scripts, although it is quite out of the league of most beginners. Beyond that, there are several commercial cross-platform frameworks, which provide interfaces for writing software on the Mac, for Windows, or even for UNIX and X/Windows. However, such frameworks are beyond the scope of this article.

Other Macintosh frameworks include MacZoop and "Yet Another Application Framework" or YAAF. Although not quite as polished as the commercial releases, these provide simple foundations suitable for most common tasks. As always, your mileage may vary.

In addition, even if you don't use a complete application framework, you may still benefit from sample frameworks for specific tasks: creating an extension, control panel, or an application plugin. The CodeWarrior CDs are chock full of such sample code and projects, and the Info-Mac archives also include many useful frameworks.

Standard developer kits, or SDKs, often provide sample code that illustrates how to perform a given task. You'll find such SDKs as part of the standard distribution of software, or available online for easy access. Of course, some software companies charge exorbitant fees for using their software SDKs, but in general most companies *want* you to help them extend their software. For example, ColorSwitch Pro is distributed with a sample framework that shows you how to write your own CSPro module.

To find more information about these frameworks and SDKs, check out:

http://www.symantec.com/scpp/fs_scpp85_mac.html
http://www.metrowerks.com/
http://devtools.apple.com/macapp/

http://mirrors.aol.com/pub/info-mac/dev/
http://mirrors.aol.com/pub/info-mac/dev/src/
http://mirrors.aol.com/pub/info-mac/dev/lib/

Application Components

There are several aspects of the Macintosh that make it a unique platform, and its important to understand these issues when programming the Macintosh.

The MacOS provides a wide range of services to applications which help simplify programming and help to maintain a consistent look and feel. These services include responding to user events, managing windows, dialogs, buttons, sliders, text, and images. These services are provided by the Toolbox, a collection of functions typically stored in the computer's ROM, and documented in the Inside Macintosh: Toolbox Essentials and Inside Macintosh: More Toolbox Essentials.

The Toolbox consists of "managers", such as the Event Manager, the Window Manager, and the Control Manager, which are composed of specific functions relating to a given task or interface element. Like most things in programming, these managers are related and often interact with each other in various ways.

Not only must a programmer learn and master a programming language, but he must also become acquainted with the Toolbox to effectively write Macintosh software. This is one reason that most programmers find it simpler to begin with a framework -- so that they can reduce the complexity of performing most tasks, and only focus on learning how to do "interesting" things.

Next, the Macintosh uses a unique mechanism for distributing application data on disk. Most systems store executable code, text, graphics, and other types of data in one or more files that are stored with the application. On the Macintosh, files are divided into 2 parts or forks: a straight collection of data bytes (like in other file systems) in the "data fork", and a structured collection of variable-length and -type data streams in the "resource fork".

The data fork usually contains traditional file data, such as the text of word processing files or lengthy binary data such as a graphic or sound file. The resource fork usually contains supplementary data, such as icons, windows, user strings, or even executable code. The advantage to structured files comes from the ability to combine multiple data streams into a single file, which is easier for the user to copy and track, and more reliable for the programmer who doesn't want to lose important data.

The MacOS Resource Manager helps manage the structured data within the resource fork. Along with the Memory Manager, application software can use it to load precompiled data, track it in memory, and release it when done.

For historical reasons, Macintosh programs must request a certain amount of memory, called a partition, so that they can run. The Memory Manager tracks an application's memory usage, and carefully shuffles internal data to maintain enough free memory for interacting with the user. Most systems provide only "nonrelocatable" memory, via traditional pointers -- once allocated, this data stays at the same memory address until it is released. The Macintosh also provides "relocatable" memory, called a Handle, which is used for large or numerous memory allocations which may fill up the fixed application partition.

Many Toolbox managers use Handles for storing internal data in memory, such as the Resource Manager and the Control Manager. This enables the system to compact relocatable memory to satisfy a large memory request, which may fail due to heap fragmentation.

Unfortunately memory management is a complicated and delicate skill, and quite beyond the scope of this introductory article. Currently, the only issue you should understand is the difference between pointers and Macintosh Handles. A future article will discuss the Memory Manager, and effective management of your
application's memory partition.

Future Directions

In the next few articles, I'll discuss writing an event loop for a graphical user interface, creating window and dialog interfaces, and drawing graphic images to the screen. These articles will walk you through the fundamentals of writing Macintosh software. As time progresses, the focus will shift to specific technologies, such as QuickTime or Game Sprockets -- something to help you make software you can be proud of.