Technical Note TN2034

Mac OS X Programming Guidelines

CONTENTS

Mac OS X is designed to be a powerful, robust, and versatile operating system. For it to live up to its full potential, however, requires adherence to some specific programming practices.

[Nov 26 2001]

Please keep the following points in mind when developing for Mac OS X:

Binaries should be Mach-O

Mach-O is the native executable format of Mac OS X. This has several implications for code that is compiled into that format, and for code that isn't. Mach-O code gets the most efficient access to all Mac OS X technologies and the best possible integration with system software. Code that isn't Mach-O doesn't.

Mach-O is supported by Apple's development tools, CodeWarrior Pro 7 and Absoft Pro Fortran for Mac OS X at the present time. Others are working on it as well.

As a developer you have several options for moving to Mach-O:

1. Use Apple Development tools:
    - Apple delivers a complete suite of development tools for Mac OS X: Mach-O code generation with C, C++ and Objective-C languages, complete Carbon, Cocoa and I/O Kit support, full Java development support, Aqua interface design tools, along with debugging and performance analysis tools. The development tools suite centers around the Project Builder IDE and Interface Builder UE design applications.
    - They are free and support all the benefits of Mac OS X like application packaging, Mach-O binary, multiple localizations, Cocoa, Carbon.
    - Interface Builder and Project Builder are available to all our developers on Apple developer's web site.
2. Use Metrowerks CodeWarrior 7:

- CodeWarrior for Mac OS, version 7 fully supports Mach-O development using Carbon or the BSD APIs in Mac OS X. CodeWarrior for Mac 7 includes a Mach-O C/C++ compiler, linker, and debugger.
- All CodeWarrior libraries, including MSL C++ and the PowerPlant C++ application framework, have been updated to build for Mach-O.
- CodeWarrior for Mac 7 includes a project conversion utility that will easily allow developers to convert their existing PEF-based Carbon applications to build as Mach-O Carbon applications.

You now have all the tools to move to Mach-O.

> **Note:**
> If you have a Carbon application that you want to run on both Mac OS X and Mac OS 9, you should package your application as a bundle that contains a CFM binary optimized for Mac OS 9 and a Mach-O binary optimized for Mac OS X.

Information about binary formats and bundle packaging is available in Inside Mac OS X: System Overview.

Back to top

Run performance tools on your binary

Once a project is functionally complete, it still requires several iterations to improve performance. For this task, you have a suite of tools at your disposal, including `top`, `MallocDebug`, `fs_usage`, `sample`, `leaks`, and `Sampler`. Use these tools to monitor activity during all phases of operation (launch, opening files, and so on). When an application is properly tuned, these tools should detect the minimal activity possible during each phase.

> **Note:**
> While the Carbon API preserves most application semantics, that does not imply that the same application on Mac OS X and Mac OS 9 will have the same performance characteristics. Certain activities are faster on Mac OS X and others are faster on Mac OS 9. A Carbon application that has been optimized for Mac OS 9 should be completely retuned for Mac OS X because the performance tradeoffs are different.

You'll find documentation on how to use performance tools Apple ships with Mac OS X at the following URL: http://developer.apple.com/techpubs/macosx/Essentials/Performance/Performance.pdf

Back to top

Don't use processor resources unless you have to

Remember that your code is executing on a fully preemptive, multitasking operating system. Thus every cycle that your code uses is not available for something else. Polling and spin loops on Mac OS 9 might have few ill effects, but on Mac OS X they have many. When the user is doing nothing with your application, make sure your application itself is doing nothing. That means it should be using zero - yes, zero - CPU cycles.

Carbon events can help you achieve this.

Back to top

Use Carbon Events in your Application

You'll find documentation on Carbon events here: http://developer.apple.com/techpubs/macosx/Carbon/oss/CarbonEventManager/carboneventmanager.html

Back to top

Avoid using resource forks

Mac OS X is intended to be an excellent Web citizen, a player in a networked world where often only "flat files" are recognized. It must provide access to file systems and network protocols such as WebDAV, NFS, and SMB.

Toward this end, the resource forks of HFS and HFS+ files should not contain resources or any other critical data. Carbon applications should put their resource data in the data fork of separate files (such as .rsrc files). This strategy also makes applications easier to internationalize.

Back to top

Use file extensions

If your application creates documents, those documents should be saved under the filename extensions claimed by the application in its Info.plist. Your application may use type and creator codes as an additional means of document typing, but extensions are essential because they are more durable. As with resource-fork data, type and creator codes (which are stored in the Finder Info fork) can be stripped off as a file travels between different file systems. Unless a user deliberately removes them, file extensions are left intact. More information here: http://developer.apple.com/techpubs/macosx/ReleaseNotes/FileExtensionGuidelines.html

Back to top

Make your code volume-format independent

Carbon pays attention to the underlying volume format and shields your application from volume format specific issues. This means that your Mac OS X application will live harmoniously in heterogeneous environments. But in turn, your application needs to be a good citizen and avoid writing out meta file information, because meta information is only natively supported on HFS and HFS+ file systems. Meta information therefore incurs a performance overhead on non-HFS(+) file systems. If your Carbon application uses the older FSSpec API's, convert to the new FSRef API's so your application works well with our volume formats.

Back to top

Use bundled resources and Unicode-savvy APIs

The success of Mac OS X depends not only on its reception in the United States but also in other countries throughout the world. For Mac OS X to be a truly internationalized operating system, applications and framework bundles should appropriately package resources and their localizations. Additionally, they should use the appropriate APIs for handling and converting Unicode text, and for providing multiscript support.

Carbon developers can find extensive documentation on Mac OS X text technologies here: http://developer.apple.com/intl/ (information on ATSUI, MLTE and Quartz text rendering as well as localization)

Back to top

Investigate using path-based file-system APIs

Paths provide the native access to files on a volume-format independent operating system like Mac OS X. All other mechanisms, such as `FSRef`s and `FSSpec`s, are built on top of paths and therefore incur some performance cost. This cost can be considerable when converting back and forth in between paths (or `CFURL`s) and `FSRef`s (or `FSSpec`s). When

developing for Mac OS X, consider using POSIX paths and `CFURL`s to represent your files, especially `CFURL`s. `CFURL` performs all translations between different file-system representations, can handle path components that are fully internationalized and of arbitrary length. Additionally, you can pass `CFURL`s around without incurring any I/O cost until the moment disk access occurs. Even if there is a good reason to use an `FSRef` for some specific `file`, consider using `CFURL` for general file access and only using an `FSRef` where there's a good user experience justification for doing so.

Back to top

Cocoa: The quickest way to developing the next killer application for Mac OS X

Based on OpenStep, Cocoa is a mature object-oriented technology that gives you basic application functionality for free, automatically supporting all Mac OS X application services. Rather than spending time "reinventing the wheel," leverage Foundation and AppKit's frameworks and devote your development to working on the extended functionality that sets your application apart from the rest.

Training classes and excellent documentation are available to help get you started with Cocoa.

Documentation for Cocoa developers can be found at http://developer.apple.com/techpubs/macosx In addition, we are working on a new tutorial for Cocoa development so please check our web site regularly.

Documentation on using Interface Builder (a great tool to prototype or make your UI for Carbon And Cocoa) can be found here: http://developer.apple.com/techpubs/macosx/DeveloperTools/devtools.html

Back to top

Be judicious using C++ for new development

Although C++ provides several attractive features, especially for application developers, experience has shown it also presents a couple of risks to be guarded against.

The statically-bound nature of C++ class libraries is generally incompatible with the dynamic code loading and software updated feature of Mac OS X. If there is any chance that your favorite collection of useful functions will ever need to be packaged into a shared library, do not use C++. Good old 'C' is the preferred language for performance-critical shared libraries. Where it's necessary to export object-oriented interfaces, both Java and Objective-C provide dynamic class loading mechanisms that work well with Mac OS X.

"Object-oriented Programming and the Objective-C Language", a book on Objective-C, can be found at http://developer.apple.com/techpubs/macosx. Java documentation is available from many different sources.

In addition to the shared library consideration, the C++ language itself presents a number of challenges to creating portable code. C++ specifies a number of language features that are not universally supported across different compilers. Exploiting all the features of C++ in your code can make porting your software to new platforms and new compilers difficult. Where you must use C++, be careful to restrict your use of the language to the absolute basics.

In addition to the specific sources of information mentioned above, there are other places to get help.

The Mac OS X System Overview provides details about many of the topics listed above, and is being updated to provide even more.

The Carbon Porting Guide and the Core Foundation documentation have valuable information for Carbon developers.

All of these documents can be found on the public Apple Developer Connection web site, at http://developer.apple.com/techpubs.

Back to top

References

Developer technotes with focused responses to specific developer issues

Developer Q&A's

Back to top

Downloadables

 Acrobat version of this Note (36K) Download

Back to top

Technical Notes by Date | Number | Technology | Title
Developer Documentation | Technical Q&As | Development Kits | Sample Code

---