



Developing for the Macintosh

White Paper

Abstract

This paper describes the process and the tools available for developers to learn to write programs for the Macintosh OS X and previous versions of Mac OS.

Tometa creates custom software for you

Tometa Software designs and develops robust software solutions for virtually all industries including in-house (vertical market) and retail software, some of which is on the shelves at your local software store. We focus our unique combination of creative, technical, and problem-solving skills on meeting our client's objectives. Because of our clarity of purpose, commitment to process, and broad professional skill sets, we are able to provide our clients with world-class solutions that are functionally superior and fully aligned with our client's strategic focus.

Balancing development speed, quality and cost is what we are all about. Tometa combines agile development practices with fixed pricing, so you know what the cost, end product, and delivery time table look like—up front. If we underestimate the effort, we complete the overrun on our dime. Simple as that. That's why large enterprise firms like Alcoa and NASDAQ choose Tometa.

Tometa's agile development expertise and low-overhead US location keep our prices comparable to offshore vendors – without offshore challenges. Using a fixed pricing model, we provide upfront visibility into a project's ultimate costs, end product and delivery schedule. Our clients like knowing that we have “skin in the game” – a fixed price that aligns our goals with yours, incenting us to get the job done right and fast.

Lastly, as a Microsoft Certified Gold Partner, Tometa Software, can customize its products or create custom web, client/server, and traditional applications. With programming experience in C#, C++, Visual Basic, CGI, HTML, RPG, Delphi, Java and many others; Tometa Software is uniquely positioned to meet your needs as a development firm.

[Check us out today](#)

Overview

At the time this report is being written, developing applications for the Macintosh is going through a period of transition. Versions of Macintosh OS previous to Mac OS X are being phased out and new applications being developed have no backwards compatibility. Apple is supporting its new rapid application development framework known as Cocoa and discontinuing support of its old development method, the Macintosh Toolbox. This makes now a great time to learn how to develop for the Macintosh platform. Since it is rare to find developers that have primarily developed for Macintosh, it is easiest to compare developing for the Macintosh with developing for the PC. Therefore, this report will assume some familiarity with developing Windows applications.

The development tools used on the PC and the Macintosh differ in their syntax and the languages used. The look and feel of Macintosh programs is somewhat different than the look and feel of Windows programs. Also, the majority of modern Macintosh programs use a native framework that is much higher-level than developing in the win32 API and MFC.

The tools Apple provides for development with the Macintosh are very strong, easy to learn, and can be used for rapid-application development. Macintosh's Cocoa, Carbon, and Classic development tools attempt to provide similar functionality to other operating systems while providing operability in a unique fashion.

Cocoa

Cocoa provides basic application framework that helps automate taking care of events in event-driven programming. It automates a lot of the tedious work a designer has to take care of when designing a GUI. It also contains a large library of standard components providing a step towards a consistent look-and-feel across the platform. Prior to OS-X version 10.4, Cocoa's GUIs could be developed using the same techniques as Swing in Java, but that support has been discontinued.

The intent of the Cocoa framework is somewhat similar to the intent of the .NET framework in application design. Both Cocoa and .NET provide the following features:

- Allows high-level rapid application development using a standard framework;
- Utilizes late object binding;
- Employs the model-view-controller (MVC) design;
- Possesses some form of automated garbage collection;
- Provides a graphic interface to assist in the design of graphical user interfaces.

However, the .NET framework attempts to provide compatibility across all versions of Windows and ideally, all computing platforms using an interpreted assembly language as opposed to Cocoa which only attempts to provide functionality in OS X by being a development method proprietary to Macintosh and creating applications to run in a native API.

The advantage of the proprietary nature of the Macintosh is that phasing-out operating systems is a less daunting task than doing it on a less proprietary scheme. Macintosh has effectively phased out versions of Mac OS prior to Mac OS X and provided a rapid application development framework called Cocoa that only works with Mac OS X. This provides full promotion of the newest features installed in Mac OS which include an enhanced GUI provider known as Quartz Compositor. Necessary steps forward like use of Unicode and enhanced localization support are also supported. The Mac OS X core is built off of UNIX which allows closer portability to the x86 architecture associated with Intel processors.

Cocoa solutions are developed using the Objective-C language. Objective-C is an object-oriented version of C. It borrows most of its syntax from C but borrows its concepts from the Smalltalk programming language. The fundamental difference between Objective-C and the objective model used in C++ or other C based languages most developers are used to, is that in Objective-C messages are sent to interact with objects instead of functions from objects being called. This creates several new ideas that are difficult to grasp for the developer accustomed to traditional C++ programming. Delegation and forwarding concepts unique to Objective-C are concepts that allow for very open-ended but very dangerous programming. Objective-C is a powerful language with a fairly steep learning curve.

Because of the nature of Objective-C there are other options that have been implemented by third party developers for Cocoa. A developer can develop in a front-end known as Objective-C++ or can even develop in a syntactical representation of C# known as Cocoa#.

Apple's XCode development environment contains the GNU Compiler Collection which allows compilation of C, C++, Java, and Objective-C using the Cocoa model. This provides familiarity to the C++-based developer; however the standard language for Cocoa development is still Objective-C and all solutions developed using Cocoa end up being bridged into Objective-C.

With Cocoa, Apple has provided an immensely powerful development studio for free on their Macintosh platform. Cocoa allows for rapid development of powerful applications and ensures that if the components of design are used correctly, that the applications created will have a standard look-and-feel. With many options in how solutions are created, Macintosh has provided a promising application

model to all developers. With the movement towards Intel's x86 architecture, an eventual bridging of Apple technology and PC technology are all but inevitable.

Conversely though, it would be foolish to force developers to use a high-level framework for all application design on a platform, so Macintosh has also provided a foundation framework to allow the developer access to the lower level components of application design. The foundation framework can also be used to develop performance based applications that do not require a GUI.

Carbon

Since Cocoa is a high-level development studio, something was needed to open up the computer to developers to create immersive graphical applications that do not use up a majority of system resources. Carbon is that solution. Carbon is Macintosh's OS level API. It allows for the most flexibility in creating applications like games and graphic manipulation programs that require the most careful optimization in how the code is executed.

The low-level nature of Carbon and its use of procedural design provide the starting point for developers looking to port their application from Windows or Linux to the Macintosh. Carbon develops applications quite similarly to both Windows and Linux so that in porting the application to the Macintosh, its quite likely code can be reused with minimal tweaking.

Since Carbon is based on procedural design, languages that developers on the PC platform are familiar with can be directly implemented. Anything supported by the GNU compiler can be compiled to run with the Carbon API. This means that Carbon is an ASCII-compatible C and C++ interface. Because of this, Carbon provides the most direct route for porting applications to the Macintosh. At the low-level Carbon is event-based programming. The large difference between Carbon and Win32 though, is that events in Carbon are handled at the event-level and events in Win32 are handled at the window-level. This is the highest priority change to learn in adapting to OS X when programming in the Carbon API.

Finally, since Carbon is based at a lower level, it allows developers to port their applications from older versions of Mac OS. OS X was not based on the previous versions of Mac OS at the system level, so programs developed for previous versions will have to under go some minimal changes if any of the new functionality of Mac OS X is required.

Classic

Macintosh still supports programs written for previous versions of Mac OS using its Classic API. Although solutions can still be developed for the Classic mode, it

would be silly to not just develop new applications in Cocoa, Carbon or even Java. However, if you are developing an application on a machine running Mac OS X, and you'd like it to be backwards compatible with previous versions of Mac OS, this is the only way to get it done.

Overview and Conclusion

The Macintosh supports a variety of API's and frameworks natively; the major two being Cocoa and Carbon. It also supports Java and a Classic API to allow direct compatibility with previous versions of the Macintosh OS and with other systems. When developing applications for the Macintosh, choosing the API can be a tough decision.

Cocoa provides high-level rapid application development in Objective-C. Cocoa provides the fastest development time for applications on the Macintosh in a method that can easily take advantage of all that Mac OS X has to offer, but because Cocoa applications are programmed in Objective-C, this makes it difficult to port them to other systems.

Carbon provides easily portability with other systems and provides more control over the system to the designer through lower-level application development in procedural based languages. Unfortunately though, It takes more time to develop a Carbon-based solution compared to a Cocoa based solution.

The Java and Classic API's have very specific uses and a developer should know right away when this is going to be the API that is needed.

Although the Macintosh supports two very capable API's, they are both very necessary to the role that Macintosh has in modern computing. While developing for the Macintosh is often a second thought in the software developers mind, Carbon provides a door for PC developers to create high-quality Macintosh applications. Cocoa, on the other hand, provides an initial development studio that is quite possibly the best available today on any platform.

This document is for informational purposes only. Tometa Software, Inc. MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

© 2004 Tometa Software, Inc. All rights reserved.