Mac OS X Technology Overview

Mac OS X

2008-10-15

Apple Inc.
© 2004, 2008 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the “keyboard” Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

.Mac is a registered service mark of Apple Inc.

Apple, the Apple logo, AirPort, AirPort Extreme, AppleScript, AppleScript Studio, AppleShare, AppleTalk, Aqua, Bonjour, Carbon, Cocoa, ColorSync, Dashcode, eMac, Final Cut, Final Cut Pro, FireWire, iBook, iCal, iChat, iTunes, Keychain, Mac, Mac OS, Macintosh, Objective-C, Pages, Quartz, QuickDraw, QuickTime, Rosetta, Safari, Sherlock, Tiger, TrueType, Velocity Engine, WebObjects, Xcode, and Xgrid are trademarks of Apple Inc., registered in the United States and other countries.

Finder, Spotlight, Time Machine, and Xserve are trademarks of Apple Inc.

Adobe, Acrobat, and PostScript are trademarks or registered trademarks of Adobe Systems Incorporated in the U.S. and/or other countries.

Intel and Intel Core are registered trademarks of Intel Corportation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

OpenGL is a registered trademark of Silicon Graphics, Inc.

PowerPC and and the PowerPC logo are trademarks of International Business Machines Corporation, used under license therefrom.

UNIX is a registered trademark of The Open Group

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED “AS IS,” AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction **Introduction to Mac OS X Technology Overview 13**

Who Should Read This Document 13
Organization of This Document 13
Getting the Xcode Tools 14
Reporting Bugs 14
See Also 15
  Developer Documentation 15
  Information on BSD 15
  Darwin and Open Source Development 16
  Other Information on the Web 16

Chapter 1 **Mac OS X System Overview 17**

A Layered Approach 17
The Advantage of Layers 18
Developer Tools 19

Chapter 2 **Darwin and Core Technologies 21**

Darwin 21
  Mach 21
  Device-Driver Support 22
  BSD 22
  File-System Support 23
  Network Support 24
  Scripting Support 28
  Threading Support 28
  X11 29
Binary File Architecture 29
  Hardware Architectures 29
  64-Bit Support 30
  Object File Formats 31
  Debug File Formats 31
  Runtime Environments 32
Security 33
IPC and Notification Mechanisms 34
  FSEvents API 34
  Kernel Queues and Kernel Events 34
  BSD Notifications 34
  Sockets, Ports, and Streams 35
  BSD Pipes 35

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Shared Memory 36
Apple Events 36
Distributed Notifications 36
Distributed Objects for Cocoa 37
Mach Messaging 37
Core Foundation 37
Objective-C 38
Java Support 38
The Java Environment 39
Java and Other Application Environments 39

Chapter 3 Graphics and Multimedia Technologies 41

Drawing Technologies 41
Quartz 41
Cocoa Drawing 43
OpenGL 44
Core Animation 44
Core Image 45
Image Kit 46
QuickDraw 46
Text and Fonts 46
Cocoa Text 47
Core Text 47
Apple Type Services 47
Apple Type Services for Unicode Imaging 48
Multilingual Text Engine 48
Audio Technologies 48
Core Audio 48
OpenAL 49
Video Technologies 49
QuickTime 50
QuickTime Kit 51
Core Video 51
DVD Playback 52
Color Management 52
Printing 52
Accelerating Your Multimedia Operations 53

Chapter 4 Application Technologies 55

Application Environments 55
Cocoa 55
Carbon 56
Java 57
AppleScript 57

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

WebObjects 58
BSD and X11 58
Application Technologies 59
Address Book Framework 59
Automator Framework 59
Bonjour 59
Calendar Store Framework 60
Core Data Framework 60
Disc Recording Framework 61
Help Support 61
Human Interface Toolbox 61
Identity Services 62
Instant Message Framework 62
Image Capture Services 63
Ink Services 63
Input Method Kit Framework 63
Keychain Services 63
Latent Semantic Mapping Services 64
Launch Services 64
Open Directory 64
PDF Kit Framework 65
Publication Subscription Framework 65
Search Kit Framework 65
Security Services 66
Speech Technologies 66
SQLite Library 67
Sync Services Framework 67
Web Kit Framework 67
Time Machine Support 68
Web Service Access 68
XML Parsing Libraries 68

Chapter 5 User Experience 71

Technologies 71
Aqua 71
Quick Look 71
Resolution-Independent User Interface 72
Spotlight 72
Bundles and Packages 73
Code Signing 73
Internationalization and Localization 74
Software Configuration 74
Fast User Switching 75
Spaces 75
Accessibility 75

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

AppleScript 76
System Applications 76
The Finder 76
The Dock 77
Dashboard 77
Automator 77
Time Machine 79

Chapter 6 Software Development Overview 81

Applications 81
Frameworks 81
Plug-ins 82
Address Book Action Plug-Ins 82
Application Plug-Ins 82
Automator Plug-Ins 83
Contextual Menu Plug-Ins 83
Core Audio Plug-Ins 83
Image Units 83
Input Method Components 84
Interface Builder Plug-Ins 84
Metadata Importers 84
QuickTime Components 85
Safari Plug-ins 85
Dashboard Widgets 85
Agent Applications 86
Screen Savers 86
Slideshows 86
Programmatic Screen Savers 87
Services 87
Preference Panes 87
Web Content 88
Dynamic Websites 88
SOAP and XML-RPC 88
Sherlock Channels 89
Mail Stationery 89
Command-Line Tools 89
Launch Items, Startup Items, and Daemons 90
Scripts 90
Scripting Additions for AppleScript 91
Kernel Extensions 92
Device Drivers 92

Chapter 7 Choosing Technologies to Match Your Design Goals 95

High Performance 95

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Easy to Use 97
Attractive Appearance 98
Reliability 99
Adaptability 100
Interoperability 101
Mobility 102

Chapter 8 **Porting Tips 103**

64-Bit Considerations 103
Windows Considerations 104
Carbon Considerations 105
  Migrating From Mac OS 9 105
  Use the Carbon Event Manager 106
  Use the HIToolbox 106
  Use Nib Files 107

Appendix A **Command Line Primer 109**

Basic Shell Concepts 109
  Getting Information 109
  Specifying Files and Directories 110
  Accessing Files on Volumes 110
  Flow Control 111
Frequently Used Commands 112
Environment Variables 113
Running Programs 114

Appendix B **Mac OS X Frameworks 115**

System Frameworks 115
  Accelerate Framework 120
  Application Services Framework 121
  Automator Framework 121
  Carbon Framework 122
  Core Services Framework 123
  Quartz Framework 123
  Web Kit Framework 124
Xcode Frameworks 124
System Libraries 125

Appendix C **Mac OS X Developer Tools 127**

Applications 127
  Xcode 127
  Interface Builder 133

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Dashcode 134
Instruments 135
Quartz Composer 136
AppleScript Studio 136
Audio Applications 137
Graphics Applications 137
Java 138
Performance Applications 139
Utility Applications 140
Command-Line Tools 143
Compiler, Linker, and Source Code Tools 144
Debugging and Tuning Tools 146
Documentation and Help Tools 149
Localization Tools 149
Version Control Tools 150
Packaging Tools 152
Scripting Tools 153
Java Tools 156
Kernel Extension Tools 157
I/O Kit Driver Tools 158

Glossary 159

Document Revision History 171

Index 173

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Figures and Tables

Chapter 1 Mac OS X System Overview 17

Figure 1-1 Layers of Mac OS X 17

Chapter 2 Darwin and Core Technologies 21

Table 2-1 Supported local volume formats 23
Table 2-2 Supported network file-sharing protocols 24
Table 2-3 Network protocols 25
Table 2-4 Network technology support 26

Chapter 3 Graphics and Multimedia Technologies 41

Figure 3-1 Quartz Compositor and the rendering APIs in Mac OS X 43
Table 3-1 Quartz technical specifications 42
Table 3-2 Partial list of formats supported by QuickTime 50
Table 3-3 Features of the Mac OS X printing system 52

Chapter 5 User Experience 71

Figure 5-1 Automator main window 78

Chapter 6 Software Development Overview 81

Table 6-1 Scripting language summary 91

Chapter 7 Choosing Technologies to Match Your Design Goals 95

Table 7-1 Technologies for improving performance 95
Table 7-2 Technologies for achieving ease of use 97
Table 7-3 Technologies for achieving an attractive appearance 98
Table 7-4 Technologies for achieving reliability 99
Table 7-5 Technologies for achieving adaptability 100
Table 7-6 Technologies for achieving interoperability 101
Table 7-7 Technologies for achieving mobility 102

Chapter 8 Porting Tips 103

Table 8-1 Required replacements for Carbon 105
Table 8-2 Recommended replacements for Carbon 106

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Appendix A Command Line Primer 109

Table A-1 Getting a list of built-in commands 109
Table A-2 Special path characters and their meaning 110
Table A-3 Input and output sources for programs 111
Table A-4 Frequently used commands and programs 112

Appendix B Mac OS X Frameworks 115

Table B-1 System frameworks 115
Table B-2 Subframeworks of the Accelerate framework 121
Table B-3 Subframeworks of the Application Services framework 121
Table B-4 Subframeworks of the Automator framework 122
Table B-5 Subframeworks of the Carbon framework 122
Table B-6 Subframeworks of the Core Services framework 123
Table B-7 Subframeworks of the Quartz framework 123
Table B-8 Subframeworks of the Web Kit framework 124
Table B-9 Xcode frameworks 124

Appendix C Mac OS X Developer Tools 127

Figure C-1 Xcode application 129
Figure C-2 Xcode documentation window 131
Figure C-3 Interface Builder 3.0 133
Figure C-4 Dashcode canvas 134
Figure C-5 The Instruments application interface 135
Figure C-6 Quartz Composer editor window 136
Figure C-7 AU Lab mixer and palettes 137
Figure C-8 iSync Plug-in Maker application 142
Figure C-9 PackageMaker application 143
Table C-1 Graphics applications 138
Table C-2 Java applications 138
Table C-3 Performance applications 139
Table C-4 CHUD applications 139
Table C-5 Utility applications 140
Table C-6 Compilers, linkers, and build tools 144
Table C-7 Tools for creating and updating libraries 145
Table C-8 Code utilities 145
Table C-9 General debugging tools 146
Table C-10 Memory debugging and tuning tools 147
Table C-11 Tools for examining code 147
Table C-12 Performance tools 148
Table C-13 Instruction trace tools 149
Table C-14 Documentation and help tools 149
Table C-15 Localization tools 150

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Table C-16 Subversion tools 150
Table C-17 RCS tools 151
Table C-18 CVS tools 151
Table C-19 Comparison tools 152
Table C-20 Packaging tools 152
Table C-21 Script interpreters and compilers 153
Table C-22 Script language converters 154
Table C-23 Perl tools 154
Table C-24 Parsers and lexical analyzers 155
Table C-25 Scripting documentation tools 155
Table C-26 Java tools 156
Table C-27 Java utilities 156
Table C-28 JAR file tools 157
Table C-29 Kernel extension tools 157
Table C-30 Driver tools 158

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Introduction to Mac OS X Technology Overview

Mac OS X is a modern operating system that combines a stable core with advanced technologies to help you deliver world-class products. The technologies in Mac OS X help you do everything from manage your data to display high-resolution graphics and multimedia content, all while delivering the consistency and ease of use that are hallmarks of the Macintosh experience. Knowing how to use these technologies can help streamline your own development process, while providing you access to key Mac OS X features.

Who Should Read This Document

Mac OS X Technology Overview is an essential guide for anyone looking to develop software for Mac OS X. It provides an overview of the technologies and tools that have an impact on the development process and provides links to relevant documents and other sources of information. You should use this document to do the following:

- Orient yourself to the Mac OS X platform.
- Learn about Mac OS X software technologies, why you might want to use them, and when.
- Learn about the development opportunities for the platform.
- Get tips and guidelines on how to move to Mac OS X from other platforms.
- Find key documents relating to the technologies you are interested in.

This document does not provide information about user-level system features or about features that have no impact on the software development process.

New developers should find this document useful for getting familiar with Mac OS X. Experienced developers can use it as a road map for exploring specific technologies and development techniques.

Organization of This Document

This document has the following chapters and appendixes:

- "Mac OS X System Overview" (page 17) provides background information for understanding the terminology and basic development environment of Mac OS X. It also provides a high-level overview of the Mac OS X system architecture.
- "Darwin and Core Technologies" (page 21) describes the technologies that comprise the Darwin environment along with other key technologies that are used throughout the system.
- "Graphics and Multimedia Technologies" (page 41) describes the graphics foundations of the system, including the technologies you use for drawing to the screen and for creating audio and video content.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

- "Application Technologies" (page 55) describes the development environments (like Carbon and Cocoa) and individual technologies (like Address Book) that you use to create your applications.
- "User Experience" (page 71) describes the technologies that your application should use to provide the best user experience for the platform. This chapter also describes some of the system technologies with which your software interacts to create that experience.
- "Software Development Overview" (page 81) describes the types of software you can create for Mac OS X and when you might use each type.
- "Choosing Technologies to Match Your Design Goals" (page 95) provides tips and guidance to help you choose the technologies that best support the design goals of your application.
- "Porting Tips" (page 103) provides starter advice for developers who are porting applications from Mac OS 9, Windows, and UNIX platforms.
- "Command Line Primer" (page 109) provides an introduction to the command-line interface for developers who have never used it before.
- "Mac OS X Frameworks" (page 115) describes the frameworks you can use to develop your software. Use this list to find specific technologies or to find when a given framework was introduced to Mac OS X.
- "Mac OS X Developer Tools" (page 127) provides an overview of the available applications and command-line tools you can use to create software for Mac OS X.

Getting the Xcode Tools

Apple provides a comprehensive suite of developer tools for creating Mac OS X software. The Xcode Tools include applications to help you design, create, debug, and optimize your software. This tools suite also includes header files, sample code, and documentation for Apple technologies. You can download the Xcode Tools from the members area of the Apple Developer Connection (ADC) website (http://connect.apple.com/). Registration is required but free.

For additional information about the tools available for working with Mac OS X and its technologies, see "Mac OS X Developer Tools" (page 127).

Reporting Bugs

If you encounter bugs in Apple software or documentation, you are encouraged to report them to Apple. You can also file enhancement requests to indicate features you would like to see in future revisions of a product or document. To file bugs or enhancement requests, go to the Bug Reporting page of the ADC website, which is at the following URL:

http://developer.apple.com/bugreporter/

You must have a valid ADC login name and password to file bugs. You can obtain a login name for free by following the instructions found on the Bug Reporting page.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

See Also

This document does not provide in-depth information on any one technology. However, it does point to relevant documents in the ADC Reference Library. References of the form "<*title*> in <*category*> Documentation" refer to documents in specific sections of the reference library.

For information about new features introduced in different versions of Mac OS X, see *What's New In Mac OS X*.

The following sections list additional sources of information about Mac OS X and its technologies.

Developer Documentation

When you install Xcode, the installer places the tools you need for development as well as sample code and developer documentation on your local hard drive. The default installation directory for Xcode is `/Developer` but in Mac OS X v10.5 and later you can specify a custom installation directory if desired. (This document uses the term <*Xcode*> to represent the root directory of your Xcode installation.) The Installer application puts developer documentation into the following locations:

- **General documentation**. Most documentation and sample code is installed in the <*Xcode*>`/Documentation/DocSets` directory. All documents are available in HTML format, which you can view from any web browser. To view the documentation, open the Xcode IDE and choose Help > Show Documentation Window.
- **Additional sample code**. Some additional sample programs are installed in <*Xcode*>`/Examples`. These samples demonstrate different tasks involving Mac OS X technologies.

You can also get the latest documentation, release notes, Tech Notes, technical Q&As, and sample code from the ADC Reference Library (http://developer.apple.com/referencelibrary). All documents are available in HTML and most are also available in PDF format.

Information on BSD

Many developers who are new to Mac OS X are also new to BSD, an essential part of the operating system's kernel environment. BSD (for Berkeley Software Distribution) is based on UNIX. Several excellent books on BSD and UNIX are available in bookstores.

You can also use the World Wide Web as a resource for information on BSD. Several organizations maintain websites with manuals, FAQs, and other sources of information on the subject. For information about related projects, see:

- Apple's Open Source page (http://developer.apple.com/opensource/)
- The FreeBSD project (http://www.freebsd.org)
- The NetBSD project (http://www.netbsd.org)
- The OpenBSD project (http://www.openbsd.org)

For more references, see the bibliography in *Kernel Programming Guide*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Darwin and Open Source Development

Apple is the first major computer company to make open source development a key part of its ongoing operating system strategy. Apple has released the source code to virtually all of the components of Darwin to the developer community and continues to update the Darwin code base to include improvements as well as security updates, bug fixes, and other important changes.

Darwin consists of the Mac OS X kernel environment, BSD libraries, and BSD command environment. For more information about Darwin and what it contains, see "Darwin" (page 21). For detailed information about the kernel environment, see *Kernel Programming Guide*.

Information about the Darwin open source efforts is available at http://developer.apple.com/darwin/ and at http://www.macosforge.org/.

Other Information on the Web

Apple maintains several websites where developers can go for general and technical information about Mac OS X.

- The Apple Macintosh products site (http://www.apple.com/mac) provides general information about Macintosh hardware and software.
- The Apple product information site (http://www.apple.com/macosx) provides information about Mac OS X.
- The ADC Reference Library (http://developer.apple.com/referencelibrary) features the same documentation that is installed with the developer tools. It also includes new and regularly updated documents as well as legacy documentation.
- The Apple Care Knowledge Base (http://www.apple.com/support/) contains technical articles, tutorials, FAQs, and other information.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Mac OS X System Overview

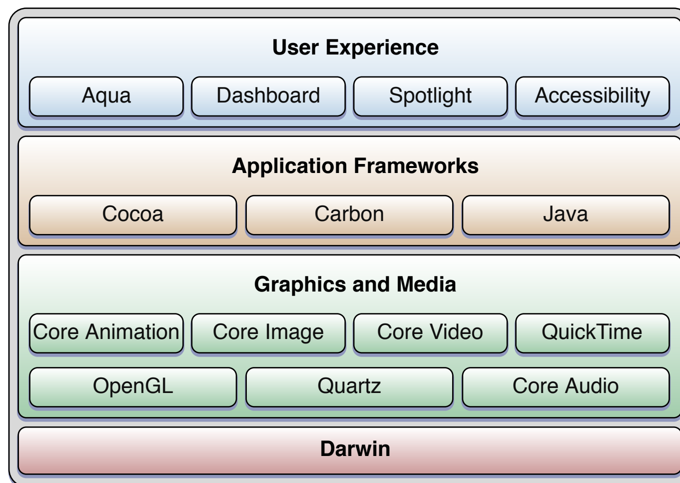
This chapter provides a high-level introduction to Mac OS X, describing its overall architecture and development tools support. The goal of this chapter is to orient you to the Mac OS X operating system and to give you a reference point from which to explore the available tools and technologies described throughout this document. Developers who are already familiar with the Mac OS X system architecture and technologies may want to skip this chapter.

Note: For a listing of commonly used Mac OS X terms, see "Glossary" (page 159).

A Layered Approach

The implementation of Mac OS X can be viewed as a set of layers. At the lower layers of the system are the fundamental services on which all software relies. Subsequent layers contain more sophisticated services and technologies that build on (or complement) the layers below. Figure 1-1 provides a graphical view of this layered approach, highlighting a few of the key technologies found in each layer of Mac OS X.

Figure 1-1 Layers of Mac OS X



The bottom layer consists of the core environment layer, of which Darwin is the most significant component. Darwin is the name given to the FreeBSD environment that comprises the heart of Mac OS X. FreeBSD is a variant of the Berkeley Software Distribution UNIX environment, which provides a secure and stable foundation for building software. Included in this layer are the kernel environment, device drivers, security support, interprocess communication support, and low-level commands and services used by all programs on the system. Besides Darwin, this layer contains several core services and technologies, many of which are simply higher-level wrappers for the data types and functions in the Darwin layer. Among the available core services

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

are those for doing collection management, data formatting, memory management, string manipulation, process management, XML parsing, stream-based I/O, and low-level network communication. For details about the technologies in this layer, see "Darwin and Core Technologies" (page 21).

The Graphics and Media layer implements specialized services for playing audio and video and for rendering 2D and 3D graphics. One of the key technologies in this layer is Quartz, which provides the main rendering environment and window management support for Mac OS X applications. QuickTime is Apple's technology for displaying video, audio, virtual reality, and other multimedia-related information. Apple's core technologies, including Core Image, Core Video, Core Animation, and Core Audio, provide advanced behavior for different types of media. OpenGL is an implementation of the industry-standard application programming interface (API) for rendering graphics and is used both as a standalone technology and as an underlying technology for accelerating all graphics operations. For details about the technologies in this layer, see "Graphics and Multimedia Technologies" (page 41).

The Application Frameworks layer embodies the technologies for building applications. At the heart of this layer are the basic environments used to develop applications: Cocoa, Carbon, Java, and others. Each environment is designed to provide a level of familiarity to certain types of developers. For example, Cocoa and Java provide object-oriented environments using the Objective-C and Java languages while Carbon provides a C-based environment. This layer also contains numerous supporting technologies, such as Core Data, Address Book, Image Services, Keychain Services, Launch Services, HTML rendering, and many others. These technologies provide advanced user features and can be used to shorten your overall development cycle. For details about the technologies in this layer, see "Application Technologies" (page 55).

The User Experience layer identifies the methodologies, technologies, and applications that make Mac OS X software unique. Apple provides countless technologies to implement the overall user experience. Many of these technologies simply work, but some require interactions with the software you create. Understanding what interactions are expected of your software can help you integrate it more smoothly into the Mac OS X ecosystem. For details about the technologies in this layer, see "User Experience" (page 71).

The Advantage of Layers

The nice thing about the Mac OS X layered design is that writing software in one layer does not preclude you from using technologies in other layers. Mac OS X technologies were built to interoperate with each other whenever possible. In cases where a given technology is unsuitable, you can always use a different technology that is suitable. For example, Cocoa applications can freely use Carbon frameworks and BSD function calls. Similarly, Carbon applications can use Objective-C based frameworks in addition to other object-oriented and C-based frameworks. Of course, in the case of Carbon, you might have to set up some Cocoa-specific structures before creating any Cocoa objects, but doing so is relatively trivial.

Although you may feel more comfortable sticking with your chosen development environment, there are advantages to straying outside of that environment. You might find that technologies in other layers offer better performance or more flexibility. For example, using the POSIX interfaces in the Darwin layer might make it easier to port your application to other platforms that conform to the POSIX specification. Having access to technologies in other layers gives you options in your development process. You can pick and choose the technologies that best suit your development needs.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Developer Tools

Mac OS X provides you with a full suite of free developer tools to prototype, compile, debug, and optimize your applications. At the heart of Apple's developer tools solution is **Xcode**, Apple's integrated development environment. You use Xcode to organize and edit your source files, compile and debug your code, view documentation, and build all manner of software products.

In addition to the Xcode application, Mac OS X also provides you with a wide selection of open source tools, such as the GNU Compiler Collection (GCC), which you use to build Mach-O programs, the native binary format of Mac OS X. If you are used to building programs from the command line, all of the familiar tools are there for you to use, including makefiles, the gdb debugger, analysis tools, performance tools, source-code management tools, and many other code utilities.

Mac OS X also provides many other tools to make the development process easier:

- **Interface Builder** lets you design your application's user interface graphically and save those designs as resource files that you can load into your program at runtime.
- **Instruments** is a powerful performance analysis and debugging tool that lets you peer into your code as it's running and gather important metrics about what it is doing.
- **Shark** is an advanced statistical analysis tool that turns your code inside out to help you find any performance bottlenecks.
- **PackageMaker** helps you build distributable packages for delivering your software to customers.
- Mac OS X includes several OpenGL tools to help you analyze the execution patterns and performance of your OpenGL rendering calls.
- Mac OS X supports various scripting languages, including Perl, Python, Ruby, and others.
- Mac OS X includes tools for creating and working with Java programs.

Installing the developer tools also installs the header files and development directories you need to develop software. For information on how to get the developer tools, see "Getting the Xcode Tools" (page 14). For more information about the tools themselves, see "Mac OS X Developer Tools" (page 127).

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Darwin and Core Technologies

This chapter summarizes the fundamental system technologies and facilities that are available to developers in Mac OS X. If you are new to developing Mac OS X software, you should read through this chapter at least once to understand the available technologies and how you might use them in your software. Even experienced developers should revisit this chapter periodically to remind themselves of the available technologies and look for recently introduced technologies.

Darwin

Beneath the appealing, easy-to-use interface of Mac OS X is a rock-solid, UNIX-based foundation called Darwin that is engineered for stability, reliability, and performance. Darwin integrates a number of technologies, the most important of which are Mach 3.0, operating-system services based on FreeBSD 5, high-performance networking facilities, and support for multiple, integrated file systems. Because the design of Darwin is highly modular, you can dynamically add such things as device drivers, networking extensions, and new file systems.

The following sections describe some of the key features of Darwin. For pointers to more information, see *Getting Started with Darwin*.

Mach

Mach is at the heart of Darwin because it provides some of the most critical functions of the operating system. Much of what Mach provides is transparent to applications. It manages processor resources such as CPU usage and memory, handles scheduling, enforces memory protection, and implements a messaging-centered infrastructure for untyped interprocess communication, both local and remote. Mach provides many important advantages to Macintosh computing:

- **Protected memory**. The stability of an operating system should not depend on all executing applications being good citizens. Even a well-behaved process can accidentally write data into the address space of the system or another process, which can result in the loss or corruption of data or even precipitate system crashes. Mach ensures that an application cannot write in another application's memory or in the operating system's memory. By walling off applications from each other and from system processes, Mach makes it virtually impossible for a single poorly behaved application to damage the rest of the system. Best of all, if an application crashes as the result of its own misbehavior, the crash affects only that application and not the rest of the system.
- **Preemptive multitasking**. With Mach, processes share the CPU efficiently. Mach watches over the computer's processor, prioritizing tasks, making sure activity levels are at the maximum, and ensuring that every task gets the resources it needs. It uses certain criteria to decide how important a task is and therefore how much time to allocate to it before giving another task its turn. Your process is not dependent on another process yielding its processing time.
- **Advanced virtual memory**. In Mac OS X, virtual memory is "on" all the time. The Mach virtual memory system gives each process its own private virtual address space. For 32-bit applications, this virtual address space is 4 GB. For 64-bit applications, the theoretical maximum is approximately 18 exabytes,

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

or 18 billion billion bytes. Mach maintains address maps that control the translation of a task's virtual addresses into physical memory. Typically only a portion of the data or code contained in a task's virtual address space resides in physical memory at any given time. As pages are needed, they are loaded into physical memory from storage. Mach augments these semantics with the abstraction of memory objects. Named memory objects enable one task (at a sufficiently low level) to map a range of memory, unmap it, and send it to another task. This capability is essential for implementing separate execution environments on the same system.

- **Real-time support**. This feature guarantees low-latency access to processor resources for time-sensitive media applications.

Mach also enables cooperative multitasking, preemptive threading, and cooperative threading.

Device-Driver Support

Darwin offers an object-oriented framework for developing device drivers called the I/O Kit framework. This framework facilitates the creation of drivers for Mac OS X and provides much of the infrastructure that they need. It is written in a restricted subset of C++. Designed to support a range of device families, the I/O Kit is both modular and extensible.

Device drivers created with the I/O Kit acquire several important features:

- True plug and play
- Dynamic device management ("hot plugging")
- Power management (for both desktops and portables)

If your device conforms to standard specifications, such as those for mice, keyboards, audio input devices, modern MIDI devices, and so on, it should just work when you plug it in. If your device doesn't conform to a published standard, you can use the I/O Kit resources to create a custom driver to meet your needs. Devices such as AGP cards, PCI and PCIe cards, scanners, and printers usually require custom drivers or other support software in order to work with Mac OS X.

For information on creating device drivers, see *I/O Kit Device Driver Design Guidelines*.

BSD

Integrated with Mach is a customized version of the Berkeley Software Distribution (BSD) operating system (currently FreeBSD 5). Darwin's implementation of BSD includes much of the POSIX API, which higher-level applications can also use to implement basic application features. BSD serves as the basis for the file systems and networking facilities of Mac OS X. In addition, it provides several programming interfaces and services, including:

- The process model (process IDs, signals, and so on)
- Basic security policies such as file permissions and user and group IDs
- Threading support (POSIX threads)
- Networking support (BSD sockets)

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

For more information about the FreeBSD operating system, go to http://www.freebsd.org/. For more information about the boot process of Mac OS X, including how it launches the daemons used to implement key BSD services, see *System Startup Programming Topics*.

File-System Support

The file-system component of Darwin is based on extensions to BSD and an enhanced Virtual File System (VFS) design. The file-system component includes the following features:

- Permissions on removable media. This feature is based on a globally unique ID registered for each connected removable device (including USB and FireWire devices) in the system.
- Access control lists (available in Mac OS X version 10.4 and later)
- URL-based volume mount, which enables users (via a Finder command) to mount such things as AppleShare and web servers
- Unified buffer cache, which consolidates the buffer cache with the virtual-memory cache
- Long filenames (255 characters or 755 bytes, based on UTF-8)
- Support for hiding filename extensions on a per-file basis
- Journaling of all file-system types to aid in data recovery after a crash

Because of its multiple application environments and the various kinds of devices it supports, Mac OS X handles file data in many standard volume formats. Table 2-1 lists the supported formats.

Table 2-1 Supported local volume formats

Volume format	Description
Mac OS Extended Format	Also called HFS (hierarchical file system) Plus, or HFS+. This is the default root and booting volume format in Mac OS X. This extended version of HFS optimizes the storage capacity of large hard disks by decreasing the minimum size of a single file.
Mac OS Standard Format	Also called hierarchical file system, or HFS. This is the volume format in Mac OS systems prior to Mac OS 8.1. HFS (like HFS+) stores resources and data in separate forks of a file and makes use of various file attributes, including type and creator codes.
UDF	Universal Disk Format, used for hard drives and optical disks, including most types of CDs and DVDs. Mac OS X v10.4 supports UDF revisions 1.02 through 1.50 (although you cannot write out Finder Info, resource forks, and other extended attributes in these revisions). Mac OS X v10.5 and later supports reading UDF revisions 1.02 through 2.60 on both block devices and most optical media, and it supports writing to block devices and to DVD-RW and DVD+RW media using UDF 2.00 through 2.50 (except for mirrored metadata partions in 2.50). You can find the UDF specification at http://www.osta.org.
ISO 9660	The standard format for CD-ROM volumes.
NTFS	The NT File System, used by Windows computers. Mac OS X can read NTFS-formatted volumes but cannot write to them.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Volume format	Description
UFS	UNIX File System is a flat (that is, single-fork) disk volume format, based on the BSD FFS (Fast File System), that is similar to the standard volume format of most UNIX operating systems; it supports POSIX file-system semantics, which are important for many server applications. Although UFS is supported in Mac OS X, its use is discouraged.
MS-DOS (FAT)	Mac OS X supports the FAT file systems used by many Windows computers. It can read and write FAT-formatted volumes.

HFS+ volumes support aliases, symbolic links, and hard links, whereas UFS volumes support symbolic links and hard links but not aliases. Although an alias and a symbolic link are both lightweight references to a file or directory elsewhere in the file system, they are semantically different in significant ways. For more information, see "Aliases and Symbolic Links" in *File System Overview*.

Note: Mac OS X does not support stacking in its file-system design.

Because Mac OS X is intended to be deployed in heterogeneous networks, it also supports several network file-sharing protocols. Table 2-2 lists these protocols.

Table 2-2 Supported network file-sharing protocols

File protocol	Description
AFP client	Apple Filing Protocol, the principal file-sharing protocol in Mac OS 9 systems (available only over TCP/IP transport).
NFS client	Network File System, the dominant file-sharing protocol in the UNIX world.
WebDAV	Web-based Distributed Authoring and Versioning, an HTTP extension that allows collaborative file management on the web.
SMB/CIFS	SMB/CIFS, a file-sharing protocol used on Windows and UNIX systems.

Network Support

Mac OS X is one of the premier platforms for computing in an interconnected world. It supports the dominant media types, protocols, and services in the industry as well as differentiated and innovative services from Apple.

The Mac OS X network protocol stack is based on BSD. The extensible architecture provided by network kernel extensions, summarized in "Networking Extensions" (page 28), facilitates the creation of modules implementing new or existing protocols that can be added to this stack.

Standard Network Protocols

Mac OS X provides built-in support for a large number of network protocols that are standard in the computing industry. Table 2-3 summarizes these protocols.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Table 2-3 Network protocols

Protocol	Description
802.1x	802.1x is a protocol for implementing port-based network access over wired or wireless LANs. It supports a wide range of authentication methods, including TLS, TTLS, LEAP, MDS, and PEAP (MSCHAPv2, MD5, GTC).
DHCP and BOOTP	The Dynamic Host Configuration Protocol and the Bootstrap Protocol automate the assignment of IP addresses in a particular network.
DNS	Domain Name Services is the standard Internet service for mapping host names to IP addresses.
FTP and SFTP	The File Transfer Protocol and Secure File Transfer Protocol are two standard means of moving files between computers on TCP/IP networks. (SFTP support was added in Mac OS X version 10.3.)
HTTP and HTTPS	The Hypertext Transport Protocol is the standard protocol for transferring webpages between a web server and browser. Mac OS X provides support for both the insecure and secure versions of the protocol.
LDAP	The Lightweight Directory Access Protocol lets users locate groups, individuals, and resources such as files and devices in a network, whether on the Internet or on a corporate intranet.
NBP	The Name Binding Protocol is used to bind processes across a network.
NTP	The Network Time Protocol is used for synchronizing client clocks.
PAP	The Printer Access Protocol is used for spooling print jobs and printing to network printers.
PPP	For dialup (modem) access, Mac OS X includes PPP (Point-to-Point Protocol). PPP support includes TCP/IP as well as the PAP and CHAP authentication protocols.
PPPoE	The Point-to-Point Protocol over Ethernet protocol provides an Ethernet-based dialup connection for broadband users.
S/MIME	The Secure MIME protocol supports encryption of email and the attachment of digital signatures to validate email addresses. (S/MIME support was added in Mac OS X version 10.3.)
SLP	Service Location Protocol is designed for the automatic discovery of resources (servers, fax machines, and so on) on an IP network.
SOAP	The Simple Object Access Protocol is a lightweight protocol for exchanging encapsulated messages over the web or other networks.
SSH	The Secure Shell protocol is a safe way to perform a remote login to another computer. Session information is encrypted to prevent unauthorized snooping of data.
TCP/IP and UDP/IP	Mac OS X provides two transmission-layer protocols, TCP (Transmission Control Protocol) and UDP (User Datagram Protocol), to work with the network-layer Internet Protocol (IP). (Mac OS X 10.2 and later includes support for IPv6 and IPSec.)

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Protocol	Description
XML-RPC	XML-RPC is a protocol for sending remote procedure calls using XML over the web.

Apple also implements a number of file-sharing protocols; see Table 2-2 (page 24) for a summary of these protocols.

Legacy Network Services and Protocols

Apple includes the following legacy network products in Mac OS X to ease the transition from earlier versions of the Mac OS.

- AppleTalk is a suite of network protocols that is standard on the Macintosh and can be integrated with other network systems. Mac OS X includes minimal support for compatibility with legacy AppleTalk environments and solutions.
- Open Transport implements industry-standard communications and network protocols as part of the I/O system. It helps developers incorporate networking services in their applications without having to worry about communication details specific to any one network.

These protocols are provided to support legacy applications, such as those running in the Classic environment. You should never use these protocols for any active development. Instead, you should use newer networking technologies such as CFNetwork.

Network Technologies

Mac OS X supports the network technologies listed in Table 2-4.

Table 2-4 Network technology support

Technology	Description
Ethernet 10/100Base-T	For the Ethernet ports built into every new Macintosh.
Ethernet 1000Base-T	Also known as Gigabit Ethernet. For data transmission over fiber-optic cable and standardized copper wiring.
Jumbo Frame	This Ethernet format uses 9 KB frames for interserver links rather than the standard 1.5 KB frame. Jumbo Frame decreases network overhead and increases the flow of server-to-server and server-to-application data. Jumbo frames are supported in Mac OS X version 10.3 and later. Systems running Mac OS X versions 10.2.4 to 10.3 can use jumbo frames only on third-party Ethernet cards that support them.
Serial	Supports modem and ISDN capabilities.
Wireless	Supports the 802.11b, 802.11g, and 802.11n wireless network technology using AirPort and AirPort Extreme.

 © 2004, 2008 Apple Inc. All Rights Reserved.

Routing and Multihoming

Mac OS X is a powerful and easy-to-use desktop operating system but can also serve as the basis for powerful server solutions. Some businesses or organizations have small networks that could benefit from the services of a router, and Mac OS X offers IP routing support for just these occasions. With IP routing, a Mac OS X computer can act as a router or even as a gateway to the Internet. The Routing Information Protocol (RIP) is used in the implementation of this feature.

Mac OS X also allows multihoming and IP aliasing. With multihoming, a computer host is physically connected to multiple data links that can be on the same or different networks. IP aliasing allows a network administrator to assign multiple IP addresses to a single network interface. Thus one computer running Mac OS X can serve multiple websites by acting as if it were multiple servers.

Zero-Configuration Networking

Introduced in Mac OS X version 10.2, Bonjour is Apple's implementation of zero-configuration networking. Bonjour enables the dynamic discovery of computer services over TCP/IP networks without the need for any complex user configuration of the associated hardware. Bonjour helps to connect computers and other electronic devices by providing a mechanism for them to advertise and browse for network-based services. See "Bonjour" (page 59) for more information.

NetBoot

NetBoot is most often used in school or lab environments where the system administrator needs to manage the configuration of multiple computers. NetBoot computers share a single System folder, which is installed on a centralized server that the system administrator controls. Users store their data in home directories on the server and have access to a common Applications folder, both of which are also commonly installed on the server.

To support NetBoot, applications must be able to run from a shared, locked volume and write a user's personal data to a different volume. Preferences and user-specific data should always be stored in the Preferences folder of the user's home directory. Users should also be asked where they want to save their data, with the user's Documents folder being the default location. Applications must also remember that multiple users may run the application simultaneously.

See Technical Note TN1151, "Creating NetBoot Server–Friendly Applications," for additional information. For information on how to write applications that support multiple simultaneous users, see *Multiple User Environments*.

Personal Web Sharing

Personal Web Sharing allows users to share information with other users on an intranet, no matter what type of computer or browser they are using. Basically, it lets users set up their own intranet site. Apache, the most popular web server on the Internet, is integrated as the system's HTTP service. The host computer on which the Personal Web Sharing server is running must be connected to a TCP/IP network.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Networking Extensions

Darwin offers kernel developers a technology for adding networking capabilities to the operating system: network kernel extensions (NKEs). The NKE facility allows you to create networking modules and even entire protocol stacks that can be dynamically loaded into the kernel and unloaded from it. NKEs also make it possible to configure protocol stacks automatically.

NKE modules have built-in capabilities for monitoring and modifying network traffic. At the data-link and network layers, they can also receive notifications of asynchronous events from device drivers, such as when there is a change in the status of a network interface.

For information on how to write an NKE, see *Network Kernel Extensions Programming Guide*.

Network Diagnostics

Introduced in Mac OS X version 10.4, network diagnostics is a way of helping the user solve network problems. Although modern networks are generally reliable, there are still times when network services may fail. Sometimes the cause of the failure is beyond the ability of the desktop user to fix, but sometimes the problem is in the way the user's computer is configured. The network diagnostics feature provides a diagnostic application to help the user locate problems and correct them.

If your application encounters a network error, you can use the new diagnostic interfaces of CFNetwork to launch the diagnostic application and attempt to solve the problem interactively. You can also choose to report diagnostic problems to the user without attempting to solve them.

For more information on using this feature, see the header files of CFNetwork.

Scripting Support

Darwin includes all of the scripting languages commonly found in UNIX-based operating systems. In addition to the scripting languages associated with command-line shells (such as `bash` and `csh`), Darwin also includes support for Perl, Python, Ruby, and others.

In Mac OS X v10.5, Darwin added support for several new scripting features. In addition to adding support for Ruby on Rails, Mac OS X also added scripting bridges to the Objective-C classes of Cocoa. These bridges let you use Cocoa classes from within your Python and Ruby scripts. For information about using these bridges, see *Ruby and Python Programming Topics for Mac OS X*.

For information about scripting tools, see "Scripting Tools" (page 153). For information on using command-line shells, see "Command Line Primer" (page 109).

Threading Support

Mac OS X provides full support for creating multiple preemptive threads of execution inside a single process. Threads let your program perform multiple tasks in parallel. For example, you might create a thread to perform some lengthy calculations in the background while a separate thread responds to user events and updates the windows in your application. Using multiple threads can often lead to significant performance improvements in your application, especially on computers with multiple CPU cores. Multithreaded programming is not without its dangers though and requires careful coordination to ensure your application's state does not get corrupted.

 | © 2004, 2008 Apple Inc. All Rights Reserved.

All user-level threads in Mac OS X are based on POSIX threads (also known as pthreads). A pthread is a lightweight wrapper around a Mach thread, which is the kernel implementation of a thread. You can use the pthreads API directly or use any of the threading packages offered by Cocoa, Carbon, or Java, all of which are implemented using pthreads. Each threading package offers a different combination of flexibility versus ease-of-use. All offer roughly the same performance, however.

For more information about threading support and guidelines on how to use threads safely, see *Threading Programming Guide*.

X11

In Mac OS X v10.3 and later, the X11 windowing system is provided as an optional installation component for the system. This windowing system is used by many UNIX applications to draw windows, controls, and other elements of graphical user interfaces. The Mac OS X implementation of X11 uses the Quartz drawing environment to give X11 windows a native Mac OS X feel. This integration also makes it possible to display X11 windows alongside windows from native applications written in Carbon and Cocoa.

Binary File Architecture

The underlying architecture of Mac OS X executables was built from the beginning with flexibility in mind. This flexibility has become important as Macintosh computers have transitioned from using PowerPC to Intel CPUs and from supporting only 32-bit applications to 64-bit applications in Mac OS X v10.5. The following sections provide an overview of the types of architectures you can support in your Mac OS X executables along with other information about the runtime and debugging environments available to you.

Hardware Architectures

When Mac OS X was first introduced, it was built to support a 32-bit PowerPC hardware architecture. With Apple's transition to Intel-based Macintosh computers, Mac OS X added initial support for 32-bit Intel hardware architectures. In addition to 32-bit support, Mac OS X v10.4 added some basic support for 64-bit architectures as well and this support was expanded in Mac OS X v10.5. This means that applications and libraries can now support four different architectures:

- 32-bit Intel (`i386`)
- 32-bit PowerPC (`ppc`)
- 64-bit Intel (`x86_64`)
- 64-bit PowerPC (`ppc64`)

Although applications can support all of these architectures in a single binary, doing so is not required. That does not mean application developers can pick a single architecture and use that alone, however. It is recommended that developers create their applications as "universal binaries" so that they run natively on both 32-bit Intel and PowerPC processors. If performance or development need warrants it, you might also add support for the 64-bit versions of each architecture.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Because libraries can be linked into multiple applications, you might consider supporting all of the available architectures when creating them. Although supporting all architectures is not required, it does give developers using your library more flexibility in how they create their applications and is recommended.

Supporting multiple architectures requires careful planning and testing of your code for each architecture. There are subtle differences from one architecture to the next that can cause problems if not accounted for in your code. For example, the PowerPC and Intel architectures use different endian structures for multi-byte data. In addition, some built-in data types have different sizes in 32-bit and 64-bit architectures. Accounting for these differences is not difficult but requires consideration to avoid coding errors.

Xcode provides integral support for creating applications that support multiple hardware architectures. For information about tools support and creating universal binaries to support both PowerPC and Intel architectures, see *Universal Binary Programming Guidelines, Second Edition*. For information about 64-bit support in Mac OS X, including links to documentation for how to make the transition, see "64-Bit Support" (page 30).

64-Bit Support

Mac OS X was initially designed to support binary files on computers using a 32-bit architecture. In Mac OS X version 10.4, however, support was introduced for compiling, linking, and debugging binaries on a 64-bit architecture. This initial support was limited to code written using C or C++ only. In addition, 64-bit binaries could link against the Accelerate framework and `libSystem.dylib` only.

In Mac OS X v10.5, most system libraries and frameworks are now 64-bit ready, meaning they can be used in both 32-bit and 64-bit applications. The conversion of frameworks to support 64-bit required some implementation changes to ensure the proper handling of 64-bit data structures; however, most of these changes should be transparent to your use of the frameworks. Building for 64-bit means you can create applications that address extremely large data sets, up to 128TB on the current Intel-based CPUs. On Intel-based Macintosh computers, some 64-bit applications may even run faster than their 32-bit equivalents because of the availability of extra processor resources in 64-bit mode.

Although most APIs support 64-bit development, some older APIs were not ported to 64-bit or offer restricted support for 64-bit applications. Many of these APIs are legacy Carbon managers that have been either wholly or partially deprecated in favor of more modern equivalents. What follows is a partial list of APIs that will not support 64-bit. For a complete description of 64-bit support in Carbon, see *64-Bit Guide for Carbon Developers*.

- Code Fragment Manager (use the Mach-O executable format instead)
- Desktop Manager (use Icon Services and Launch Services instead)
- Display Manager (use Quartz Services instead)
- QuickDraw (use Quartz or Cocoa instead)
- QuickTime Musical Instruments (use Core Audio instead)
- Sound Manager (use Core Audio instead)

In addition to the list of deprecated APIs, there are a few modern APIs that are not deprecated, but which have not been ported to 64-bit. Development of 32-bit applications with these APIs is still supported, but if you want to create a 64-bit application, you must use alternative technologies. Among these APIs are the following:

- The entire QuickTime C API (not deprecated, but developers should use QuickTime Kit instead in 64-bit applications)

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

- HIToolbox, Window Manager, and most other Carbon user interface APIs (not deprecated, but developers should use Cocoa user interface classes and other alternatives); see *64-Bit Guide for Carbon Developers* for the list of specific APIs and transition paths.

Mac OS X uses the LP64 model that is in use by other 64-bit UNIX systems, which means fewer headaches when porting from other operating systems. For general information on the LP64 model and how to write 64-bit applications, see *64-Bit Transition Guide*. For Cocoa-specific transition information, see *64-Bit Transition Guide for Cocoa*. For Carbon-specific transition information, see *64-Bit Guide for Carbon Developers*.

Object File Formats

Mac OS X is capable of loading object files that use several different object-file formats, including the following:

- Mach-O
- Java bytecode
- Preferred Executable Format (PEF)

Of these formats, the Mach-O format is the format used for all native Mac OS X application development. The Java bytecode format is a format executed through the Hotspot Java virtual machine and used exclusively for Java-based programs. The PEF format is handled by the Code Fragment Manager and is a legacy format that was used for transitioning Mac OS 9 applications to Mac OS X.

For information about the Mach-O file format, see *Mac OS X ABI Mach-O File Format Reference*. For additional information about using Mach-O files, see *Mach-O Programming Topics*. For information about Java support in Mac OS X, see "Java Support" (page 38). For information about the PEF format and Code Fragment Manager, see "CFM Runtime Environment" (page 32)

Debug File Formats

Whenever you debug an executable file, the debugger uses symbol information generated by the compiler to associate user-readable names with the procedure and data address it finds in memory. Normally, this user-readable information is not needed by a running program and is stripped out (or never generated) by the compiler to save space in the resulting binary file. For debugging, however, this information is very important to be able to understand what the program is doing.

Mac OS X supports two different debug file formats for compiled executables: stabs and DWARF. The stabs format is present in all versions of Mac OS X and until the introduction of Xcode 2.4 was the default debugging format. Code compiled with Xcode 2.4 and later uses the DWARF debugging format by default. When using the stabs format, debugging symbols, like other symbols are stored in the symbol table of the executable; see *Mac OS X ABI Mach-O File Format Reference*. With the DWARF format, however, debugging symbols are stored either in a specialized segment of the executable or in a separate debug-information file.

For information about the DWARF standard, go to http://www.dwarfstd.org. For information about the stabs debug file format, see *STABS Debug Format*. For additional information about Mach-O files and their stored symbols, see *Mach-O Programming Topics*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Runtime Environments

Since its first release, Mac OS X has supported several different environments for running applications. The most prominent of these environments is the Dyld environment, which is also the only environment supported for active development. Most of the other environments provided legacy support during the transition from Mac OS 9 to Mac OS X and are no longer supported for active development. The following sections describe the runtime environments you may encounter in various versions of Mac OS X.

Dyld Runtime Environment

The dyld runtime environment is the native environment in Mac OS X and is used to load, link, and execute Mach-O files. At the heart of this environment is the `dyld` dynamic loader program, which handles the loading of a program's code modules and associated dynamic libraries, resolves any dependencies between those libraries and modules, and begins the execution of the program.

Upon loading a program's code modules, the dynamic loader performs the minimal amount of symbol binding needed to launch your program and get it running. This binding process involves resolving links to external libraries and loading them as their symbols are used. The dynamic loader takes a lazy approach to binding individual symbols, doing so only as they are used by your code. Symbols in your code can be strongly-linked or weakly-linked. Strongly-linked symbols cause the dynamic loader to terminate your program if the library containing the symbol cannot be found or the symbol is not present in the library. Weakly-linked symbols terminate your program only if the symbol is not present and an attempt is made to use it.

For more information about the dynamic loader program, see `dyld`. For information about building and working with Mach-O executable files, see *Mach-O Programming Topics*.

Java Runtime Environment

The Java runtime environment consists of the HotSpot Java virtual machine, the "just-in-time" (JIT) bytecode compiler, and code packages containing the standard Java classes. For more information about Java support in Mac OS X, see "Java Support" (page 38).

CFM Runtime Environment

The Code Fragment Manager (CFM) runtime environment is a legacy environment inherited from Mac OS 9. Mac OS X provides this environment to support applications that want to use the modern features of Mac OS X but have not yet been converted over to the dyld environment for various reasons. The CFM runtime environment expects code modules to be built using the Preferred Executable Format (PEF).

Unlike the dyld environment, the CFM runtime environment takes a static approach to symbol binding. At runtime, the CFM library manager binds all referenced symbols when the code modules are first loaded into memory. This binding occurs regardless of whether those symbols are actually used during the program's course of execution. If a particular symbol is missing, the program does not launch. (An exception to this rule occurs when code modules are bound together using weak linking, which explicitly permits symbols to be missing as long as they are never used.)

Because all system libraries are implemented using Mach-O and dyld, Mac OS X provides a set of libraries to bridge calls between CFM code and system libraries. This bridging is transparent but incurs a small amount of overhead for CFM-based programs. The Carbon library is one example of a bridged library.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Note: The libraries bridge only from CFM to dyld; they do not bridge calls going in the opposite direction. It is possible for a dyld-based application to make calls into a CFM-based library using the CFBundle facility, but this solution is not appropriate for all situations. If you want a library to be available to all Mac OS X execution environments, build it as a dyld-based library.

On Intel-based Macintosh computers, CFM binaries are run under the Rosetta environment.

The Classic Environment

The Classic compatibility environment (or simply, Classic environment) is called a "software compatibility" environment because it enabled Mac OS X to run applications built for Mac OS 9.1 or 9.2. The Classic environment is not an emulator; it is a hardware abstraction layer between an installed Mac OS 9 System Folder and the Mac OS X kernel environment. Because of architectural differences, applications running in the Classic environment do not share the full advantages of the kernel environment.

The Classic environment is supported only on PowerPC-based Macintosh computers and is deprecated in Mac OS X v10.5 and later. You should not be doing any active development using the Classic environment. If you want to write programs to run in Mac OS X, you should use the dyld environment instead.

The Classic environment is not supported on Intel-based Macintosh computers.

Security

The roots of Mac OS X in the UNIX operating system provide a robust and secure computing environment whose track record extends back many decades. Mac OS X security services are built on top of two open-source standards: BSD (Berkeley Software Distribution) and CDSA (Common Data Security Architecture). BSD is a form of the UNIX operating system that provides basic security for fundamental services, such as file and network access. CDSA provides a much wider array of security services, including finer-grained access permissions, authentication of users' identities, encryption, and secure data storage. Although CDSA has its own standard API, it is complex and does not follow standard Macintosh programming conventions. Therefore, Mac OS X includes its own security APIs that call through to the CDSA API for you.

In Mac OS X v10.5 several improvements were made to the underlying operating system security, including the addition of the following features:

- Adoption of the Mandatory Access Control (MAC) framework, which provides a fine-grained security architecture for controlling the execution of processes at the kernel level. This feature enables the "sandboxing" of applications, which lets you limit the access of a given application to only those features you designate.
- Support for code signing and installer package signing. This feature lets the system validate applications using a digital signature and warn the user if an application is tampered with.
- Compiler support for fortifying your source code against potential security threats. This support includes options to disallow the execution of code located on the stack or other portions of memory containing data. It also includes some new GCC compiler warnings.
- Support for putting unknown files into quarantine. This is especially useful for developers of web browsers or other network-based applications that receive files from unknown sources. The system prevents access to quarantined files unless the user explicitly approves that access.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

For an introduction to Mac OS X security features, see *Security Overview*.

IPC and Notification Mechanisms

Mac OS X supports numerous technologies for interprocess communication (IPC) and for delivering notifications across the system. The following sections describe the available technologies.

FSEvents API

Introduced in Mac OS X v10.5, the FSEvents API notifies your application when changes occur in the file system. You can use file system events to monitor directories for any changes, such as the creation, modification, or removal of contained files and directories. Although kqueues provide similar behavior, the FSEvents API provides a much simpler way to monitor many directories at once. For example, you can use file system events to monitor entire file system hierarchies rooted at a specific directory and still receive notifications about individual directories in the hierarchy. The implementation of file system events is lightweight and efficient, providing built-in coalescing when multiple changes occur within a short period of time to one or many directories.

The FSEvents API is not intended for detecting fine-grained changes to individual files. You would not use this to detect changes to an individual file as in a virus checker program. Instead, you might use FSEvents to detect general changes to a file hierarchy. For example, you might use this technology in backup software to detect what files changed. You might also use it to monitor a set of data files your application uses, but which can be modified by other applications as well.

For information on how to use the FSEvents API, see *File System Events Programming Guide*.

Kernel Queues and Kernel Events

Kernel queues (also known as kqueues) and kernel events (also known as kevents) are an extremely powerful technology you use to intercept kernel-level events. Although often used to detect file-system changes, you can also use this technology to receive notifications about changes to sockets, processes, and other aspects of the system. For example, you could use them to detect when a process exits or when it issues `fork` and `exec` calls. Kernel queues and events are part of the FreeBSD layer of the operating system and are described in the `kqueue` and `kevent` man pages.

BSD Notifications

Starting with Mac OS X version 10.3, applications can take advantage of a system-level notification API. This notification mechanism is defined in the `/usr/include/notify.h` system header. BSD notifications offer some advantages over the Core Foundation notification mechanism, including the following:

- Clients can receive BSD notifications through several different mechanisms, including Mach ports, signals, and file descriptors.
- BSD notifications are more lightweight and efficient than other notification techniques.
- BSD notifications can be coalesced if multiple notifications are received in quick succession.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

You can add support for BSD notifications to any type of program, including Carbon and Cocoa applications. For more information, see *Mac OS X Notification Overview* or the `notify` man page.

Sockets, Ports, and Streams

Sockets and ports provide a portable mechanism for communicating between applications in Mac OS X. A socket represents one end of a communications channel between two processes either locally or across the network. A port is a channel between processes or threads on the local computer. Applications can set up sockets and ports to implement fast, efficient messaging between processes.

The Core Foundation framework includes abstractions for sockets (CFSocket/CFRunLoop) and ports (CFMessagePort). You can use CFSocket with CFRunLoop to multiplex data received from a socket with data received from other sources. This allows you to keep the number of threads in your application to an absolute minimum, which conserves system resources and thus aids performance. Core Foundation sockets are also much simpler to use than the raw socket interfaces provided by BSD. CFMessagePort provides similar features for ports.

If you are communicating using an established transport mechanism such as Bonjour or HTTP, a better way to transfer data between processes is with the Core Foundation or Cocoa stream interfaces. These interfaces work with CFNetwork to provide a stream-based way to read and write network data. Like sockets, streams and CFNetwork were designed with run loops in mind and operate efficiently in that environment.

CFSocket and its related functions are documented in *CFSocket Reference*. For information about Core Foundation streams, see *CFReadStream Reference* and *CFWriteStream Reference*. For information about Cocoa streams, see the description of the NSStream class in *Foundation Framework Reference*.

BSD Pipes

A pipe is a communications channel typically created between a parent and a child process when the child process is forked. Data written to a pipe is buffered and read in first-in, first-out (FIFO) order. You create unnamed pipes between a parent and child using the `pipe` function declared in `/usr/include/unistd.h`. This is the simplest way to create a pipe between two processes; the processes must, however, be related.

You can also create named pipes to communicate between any two processes. A named pipe is represented by a file in the file system called a FIFO special file. A named pipe must be created with a unique name known to both the sending and the receiving process.

Note: Make sure you give your named pipes appropriate names to avoid unwanted collisions caused by the presence of multiple simultaneous users.

Pipes are a convenient and efficient way to create a communications channel between related processes. However, in general use, pipes are still not as efficient as using CFStream. The run loop support offered by CFStream makes it a better choice when you have multiple connections or plan to maintain an open channel for an extended period of time.

The interfaces for CFStream are documented in *CFNetwork Programming Guide*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Shared Memory

Shared memory is a region of memory that has been allocated by a process specifically for the purpose of being readable and possibly writable among several processes. You create regions of shared memory in several different ways. Among the available options are the functions in /usr/include/sys/shm.h, the shm_open and shm_unlink routines, and the mmap routine. Access to shared memory is controlled through POSIX semaphores, which implement a kind of locking mechanism. Shared memory has some distinct advantages over other forms of interprocess communication:

- Any process with appropriate permissions can read or write a shared memory region.
- Data is never copied. Each process reads the shared memory directly.
- Shared memory offers excellent performance.

The disadvantage of shared memory is that it is very fragile. When a data structure in a shared memory region becomes corrupt, all processes that refer to the data structure are affected. In most cases, shared memory regions should also be isolated to a single user session to prevent security issues. For these reasons, shared memory is best used only as a repository for raw data (such as pixels or audio), with the controlling data structures accessed through more conventional interprocess communication.

For information about shm_open, shm_unlink, and mmap, see the shm_open, shm_unlink, and mmap man pages.

Apple Events

An Apple event is a high-level semantic event that an application can send to itself, to other applications on the same computer, or to applications on a remote computer. Apple events are the primary technology used for scripting and interapplication communication in Mac OS X. Applications can use Apple events to request services and information from other applications. To supply services, you define objects in your application that can be accessed using Apple events and then provide Apple event handlers to respond to requests for those objects.

Apple events have a well-defined data structure that supports extensible, hierarchical data types. To make it easier for scripters and other developers to access it, your application should generally support the standard set of events defined by Apple. If you want to support additional features not covered by the standard suite, you can also define custom events as needed.

Apple events are part of the Application Services umbrella framework. For information on how to use Apple events, see *Apple Events Programming Guide*. See also *Apple Event Manager Reference* for information about the functions and constants used to create, send, and receive Apple events.

Distributed Notifications

A distributed notification is a message posted by any process to a per-computer notification center, which in turn broadcasts the message to any processes interested in receiving it. Included with the notification is the ID of the sender and an optional dictionary containing additional information. The distributed notification mechanism is implemented by the Core Foundation CFNotificationCenter object and by the Cocoa NSDistributedNotificationCenter class.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Distributed notifications are ideal for simple notification-type events. For example, a notification might communicate the status of a certain piece of hardware, such as the network interface or a typesetting machine. However, notifications should not be used to communicate critical information to a specific process. Although Mac OS X makes every effort possible, it does not guarantee the delivery of a notification to every registered receiver.

Distributed notifications are true notifications because there is no opportunity for the receiver to reply to them. There is also no way to restrict the set of processes that receive a distributed notification. Any process that registers for a given notification may receive it. Because distributed notifications use a string for the unique registration key, there is also a potential for namespace conflicts.

For information on Core Foundation support for distributed notifications, see *CFNotificationCenter Reference*. For information about Cocoa support for distributed notifications, see *Notification Programming Topics for Cocoa*.

Distributed Objects for Cocoa

Cocoa distributed objects provide a transparent mechanism that allows different applications (or threads in the same application) to communicate on the same computer or across the network. The implementation of distributed objects lets you focus on the data being transferred rather than the connection. As a result, implementing distributed objects takes less time than most other IPC mechanisms; however, this ease of implementation comes at the cost of performance. Distributed objects are typically not as efficient as many other techniques.

For information on how to use distributed objects in your Cocoa application, see *Distributed Objects Programming Topics*.

Mach Messaging

Mach port objects implement a standard, safe, and efficient construct for transferring messages between processes. Despite these benefits, messaging with Mach port objects is the least desirable way to communicate between processes. Mach port messaging relies on knowledge of the kernel interfaces, which may change in a future version of Mac OS X.

All other interprocess communications mechanisms in Mac OS X are implemented using Mach ports at some level. As a result, low-level technologies such as sockets, ports, and streams all offer efficient and reliable ways to communicate with other processes. The only time you might consider using Mach ports directly is if you are writing software that runs in the kernel.

Core Foundation

The Core Foundation framework (`CoreFoundation.framework`) is a set of C-based interfaces that provide basic data management features for Mac OS X programs. Among the data types you can manipulate with Core Foundation are the following:

- Collections
- Bundles and plug-ins

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

- Strings
- Raw data blocks
- Dates and times
- Preferences
- Streams
- URLs
- XML data
- Locale information
- Run loops
- Ports and sockets

Although it is C-based, the design of the Core Foundation interfaces is more object-oriented than C. As a result, the opaque types you create with Core Foundation interfaces operate seamlessly with the Cocoa Foundation interfaces. Core Foundation is used extensively in Mac OS X to represent fundamental types of data, and its use in Carbon and other non-Cocoa applications is highly recommended. (For Cocoa applications, use the Cocoa Foundation framework instead.)

For an overview of Core Foundation, see *Core Foundation Design Concepts*. For additional conceptual and reference material, see the categories of Reference Library > Core Foundation.

Objective-C

Objective-C is a C-based programming language with object-oriented extensions. It is also the primary development language for Cocoa applications. Unlike C++ and some other object-oriented languages, Objective-C comes with its own dynamic runtime environment. This runtime environment makes it much easier to extend the behavior of code at runtime without having access to the original source.

In Mac OS X v10.5, an update to the Objective-C language (called Objective-C 2.0) was introduced, adding support for the following features:

- Object properties, which offer an alternative way to declare member variables
- Support for garbage collection; see *Garbage Collection Programming Guide*
- A new `for` operator syntax for performing fast enumerations of collections
- Protocol enhancements
- Deprecation syntax

For information about the Objective-C language, see *The Objective-C 2.0 Programming Language*.

Java Support

The following sections outline the support provided by Mac OS X for creating Java-based programs.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Note: The developer documentation on the Apple website contains an entire section devoted to Java. There you can find detailed information on the Java environment and accompanying technologies for operating in Mac OS X. For an introduction to the Java environment and pointers to relevant documentation on Java programming in Mac OS X, see *Getting Started with Java*.

The Java Environment

The libraries, JAR files, and executables for the Java application environment are located in the `/System/Library/Frameworks/JavaVM.framework` directory. The Java application environment has three major components:

- A development environment, comprising the Java compiler (`javac`) and debugger (`jdb`) as well as other tools, including `javap`, `javadoc`, and `appletviewer`. You can also build Java applications using Xcode.
- A runtime environment consisting of Sun's high-performance HotSpot Java virtual machine, the "just-in-time" (JIT) bytecode compiler, and several basic packages, including `java.lang`, `java.util`, `java.io`, and `java.net`.
- An application framework containing the classes necessary for building a Java application. This framework contains the Abstract Windowing Toolkit (`java.awt`) and Swing (`javax.swing`) packages, among others. These packages provide user interface components, basic drawing capabilities, a layout manager, and an event-handling mechanism.

Like Carbon and Cocoa applications, a Java application can be distributed as a double-clickable bundle. The Jar Bundler tool takes your Java packages and produces a Mac OS X bundle. This tool is installed along with Xcode and the rest of the Apple developer tools on the Xcode Tools CD.

If you want to run your Java application from the command line, you can use the `java` command. To launch a Java application from another program, use the system `exec` call or the Java `Runtime.exec` method. To run applets, embed the applet into an HTML page and open the page in Safari.

Java and Other Application Environments

Java applications can take advantage of Mac OS X technologies such as Cocoa and QuickTime through Sun's Java Native Interface (JNI). For details on using the JNI on Mac OS X, see *Technical Note 2147*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Graphics and Multimedia Technologies

The graphics and multimedia capabilities of Mac OS X set it apart from other operating systems. Mac OS X is built on a modern foundation that includes support for advanced compositing operations with support for hardware-based rendering on supported graphics hardware. On top of this core are an array of technologies that provide support for drawing 2D, 3D, and video-based content. The system also provides an advanced audio system for the generation, playback, and manipulation of multichannel audio.

Drawing Technologies

Mac OS X includes numerous technologies for rendering 2D and 3D content and for animating that content dynamically at runtime.

Quartz

Quartz is at the heart of the Mac OS X graphics and windowing environment. Quartz provides rendering support for 2D content and combines a rich imaging model with on-the-fly rendering, compositing, and anti-aliasing of content. It also implements the windowing system for Mac OS X and provides low-level services such as event routing and cursor management.

Quartz comprises both a client API (Quartz 2D) and a window server (Quartz Compositor). The client API provides commands for managing the graphics context and for drawing primitive shapes, images, text, and other content. The window server manages the display and device driver environment and provides essential services to clients, including basic window management, event routing, and cursor management behaviors.

The Quartz 2D client API is implemented as part of the Application Services umbrella framework (`ApplicationServices.framework`), which is what you include in your projects when you want to use Quartz. This umbrella framework includes the Core Graphics framework (`CoreGraphics.framework`), which defines the Quartz 2D interfaces, types, and constants you use in your applications.

The Quartz Services API (which is also part of the Core Graphics framework) provides direct access to some low-level features of the window server. You can use this API to get information about the currently connected display hardware, capture a display for exclusive use, or adjust display attributes, such as its resolution, pixel depth, and refresh rate. Quartz Services also provides some support for operating a Mac OS X system remotely.

For information about the Quartz 2D API, see *Quartz 2D Programming Guide*. For information about the Quartz Services API, see *Quartz Display Services Programming Topics*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Digital Paper Metaphor

The Quartz imaging architecture is based on a digital paper metaphor. In this case, the digital paper is PDF, which is also the internal model used by Quartz to store rendered content. Content stored in this medium has a very high fidelity and can be reproduced on many different types of devices, including displays, printers, and fax machines. This content can also be written to a PDF file and viewed by any number of applications that display the PDF format.

The PDF model gives application developers much more control over the final appearance of their content. PDF takes into account the application's choice of color space, fonts, image compression, and resolution. Vector artwork can be scaled and manipulated during rendering to implement unique effects, such as those that occur when the system transitions between users with the fast user switching feature.

Mac OS X also takes advantage of the flexibility of PDF in implementing some system features. For example, in addition to printing, the standard printing dialogs offer options to save a document as PDF, preview the document before printing, or transmit the document using a fax machine. The PDF used for all of these operations comes from the same source: the pages formatted for printing by the application's rendering code. The only difference is the device to which that content is sent.

Quartz 2D Features

Quartz 2D provides many important features to user applications, including the following:

- High-quality rendering on the screen
- Resolution independent UI support
- Anti-aliasing for all graphics and text
- Support for adding transparency information to windows
- Internal compression of data
- A consistent feature set for all printers
- Automatic PDF generation and support for printing, faxing, and saving as PDF
- Color management through ColorSync

Table 3-1 describes some of technical specifications for Quartz.

Table 3-1 Quartz technical specifications

Bit depth	A minimum bit depth of 16 bits for typical users. An 8-bit depth in full-screen mode is available for Classic applications, games, and other multimedia applications.
Minimum resolution	Supports 800 pixels by 600 pixels as the minimum screen resolution for typical users. A resolution of 640 x 480 is available for the iBook as well as for Classic applications, games, and other multimedia applications.
Velocity Engine and SSE support	Quartz takes advantage of any available vector unit hardware to boost performance.
Quartz Extreme	Quartz Extreme uses OpenGL to draw the entire Mac OS X desktop. Graphics calls render in supported video hardware, freeing up the CPU for other tasks.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Quartz Compositor

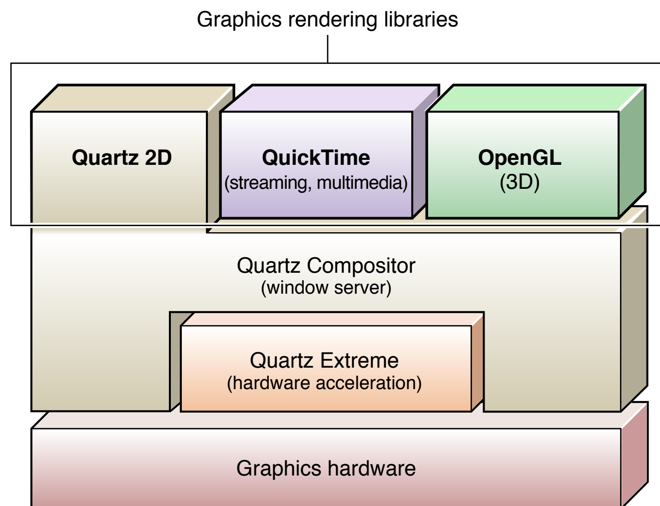
Quartz Compositor, the window server for Mac OS X, coordinates all of the low-level windowing behavior and enforces a fundamental uniformity in what appears on the screen. It manages the displays available on the user's system, interacting with the necessary device drivers. It also provides window management, event-routing, and cursor management behaviors.

In addition to window management, Quartz Compositor handles the compositing of all visible content on the user's desktop. It supports transparency effects through the use of alpha channel information, which makes it possible to display drop shadows, cutouts, and other effects that add a more realistic and dimensional texture to the windows.

The performance of Quartz Compositor remains consistently high because of several factors. To improve window redrawing performance, Quartz Compositor supports buffered windows and the layered compositing of windows and window content. Thus, windows that are hidden behind opaque content are never composited. Quartz Compositor also incorporates Quartz Extreme, which speeds up rendering calls by handing them off to graphics hardware whenever possible.

Figure 3-1 shows the high-level relationships between Quartz Compositor and the rendering technologies available on Mac OS X. QuickTime and OpenGL have fewer dependencies on Quartz Compositor because they implement their own versions of certain windowing capabilities.

Figure 3-1 Quartz Compositor and the rendering APIs in Mac OS X



Cocoa Drawing

The Cocoa application environment provides object-oriented wrappers for many of the features found in Quartz. Cocoa provides support for drawing primitive shapes such as lines, rectangles, ovals, arcs, and Bezier paths. It supports drawing in both standard and custom color spaces and it supports content manipulations using graphics transforms. Because it is built on top of Quartz, drawing calls made from Cocoa are composited along with all other Quartz 2D content. You can even mix Quartz drawing calls (and drawing calls from other system graphics technologies) with Cocoa calls in your code if you wish.

For more information on how to draw using Cocoa, see *Cocoa Drawing Guide*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

OpenGL

OpenGL is an industry-wide standard for developing portable three-dimensional (3D) graphics applications. It is specifically designed for games, animation, CAD/CAM, medical imaging, and other applications that need a rich, robust framework for visualizing shapes in two and three dimensions. The OpenGL API is one of the most widely adopted graphics API standards, which makes code written for OpenGL portable and consistent across platforms. The OpenGL framework (`OpenGL.framework`) in Mac OS X includes a highly optimized implementation of the OpenGL libraries that provides high-quality graphics at a consistently high level of performance.

OpenGL offers a broad and powerful set of imaging functions, including texture mapping, hidden surface removal, alpha blending (transparency), anti-aliasing, pixel operations, viewing and modeling transformations, atmospheric effects (fog, smoke, and haze), and other special effects. Each OpenGL command directs a drawing action or causes a special effect, and developers can create lists of these commands for repetitive effects. Although OpenGL is largely independent of the windowing characteristics of each operating system, the standard defines special glue routines to enable OpenGL to work in an operating system's windowing environment. The Mac OS X implementation of OpenGL implements these glue routines to enable operation with the Quartz Compositor.

In Mac OS X v10.5 and later, OpenGL supports the ability to use multiple threads to process graphics data. OpenGL also supports pixel buffer objects, color managed texture images in the sRGB color space, support for 64-bit addressing, and improvements in the shader programming API. You can also attach an AGL context to `WindowRef` and `HIView` objects and thereby avoid using QuickDraw ports.

For information about using OpenGL in Mac OS X, see *OpenGL Programming Guide for Mac OS X*.

Core Animation

Introduced in Mac OS X v10.5, Core Animation is a set of Objective-C classes for doing sophisticated 2D rendering and animation. Using Core Animation, you can create everything from basic window content to Front Row–style user interfaces, and achieve respectable animation performance, without having to tune your code using OpenGL or other low-level drawing routines. This performance is achieved using server-side content caching, which restricts the compositing operations performed by the server to only those parts of a view or window whose contents actually changed.

At the heart of the Core Animation programming model are layer objects, which are similar in many ways to Cocoa views. Like views, you can arrange layers in hierarchies, change their size and position, and tell them to draw themselves. Unlike views, layers do not support event-handling, accessibility, or drag and drop. You can also manipulate the layout of layers in more ways than traditional Cocoa views. In addition to positioning layers using a layout manager, you can apply 3D transforms to layers to rotate, scale, skew, or translate them in relation to their parent layer.

Layer content can be animated implicitly or explicitly depending on the actions you take. Modifying specific properties of a layer, such as its geometry, visual attributes, or children, typically triggers an implicit animation to transition from the old state to the new state of the property. For example, adding a child layer triggers an animation that causes the child layer to fade gradually into view. You can also trigger animations explicitly in a layer by modifying its transformation matrix.

You can manipulate layers independent of, or in conjunction with, the views and windows of your application. Both Cocoa and Carbon applications can take advantage of the Core Animation's integration with the `NSView` class to add animation effects to windows. Layers can also support the following types of content:

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

- Quartz Composer compositions
- OpenGL content
- Core Image filter effects
- Quartz and Cocoa drawing content
- QuickTime playback and capture

The Core Animation features are part of the Quartz Core framework (`QuartzCore.framework`). For information about Core Animation, see *Animation Overview*.

Core Image

Introduced in Mac OS X version 10.4, Core Image extends the basic graphics capabilities of the system to provide a framework for implementing complex visual behaviors in your application. Core Image uses GPU-based acceleration and 32-bit floating-point support to provide fast image processing and pixel-level accurate content. The plug-in based architecture lets you expand the capabilities of Core Image through the creation of image units, which implement the desired visual effects.

Core Image includes built-in image units that allow you to:

- Crop images
- Correct color, including perform white-point adjustments
- Apply color effects, such as sepia tone
- Blur or sharpen images
- Composite images
- Warp the geometry of an image by applying an affine transform or a displacement effect
- Generate color, checkerboard patterns, Gaussian gradients, and other pattern images
- Add transition effects to images or video
- Provide real-time control, such as color adjustment and support for sports, vivid, and other video modes
- Apply linear lighting effects, such as spotlight effects

You define custom image units using the classes of the Core Image framework. You can use both the built-in and custom image units in your application to implement special effects and perform other types of image manipulations. Image units take full advantage of hardware vector units, Quartz, OpenGL, and QuickTime to optimize the processing of video and image data. Rasterization of the data is ultimately handled by OpenGL, which takes advantage of graphics hardware acceleration whenever it is available.

Core Image is part of the Quartz Core framework (`QuartzCore.framework`). For information about how to use Core Image or how to write custom image units, see *Core Image Programming Guide* and *Core Image Reference Collection*. For information about the built-in filters in Core Image, see *Core Image Filter Reference*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Image Kit

Introduced in Mac OS X v10.5, the Image Kit framework is an Objective-C framework that makes it easy to incorporate powerful imaging services into your applications. This framework takes advantage of features in Quartz, Core Image, OpenGL, and Core Animation to provide an advanced and highly optimized development path for implementing the following features:

- Displaying images
- Rotating, cropping, and performing other image-editing operations
- Browsing for images
- Taking pictures using the built-in picture taker panel
- Displaying slideshows
- Browsing for Core Image filters
- Displaying custom views for Core Image filters

The Image Kit framework is included as a subframework of the Quartz framework (`Quartz.framework`). For more information on how to use Image Kit, see *Image Kit Programming Guide* and *Image Kit Reference Collection*

QuickDraw

QuickDraw is a legacy technology adapted from earlier versions of the Mac OS that lets you construct, manipulate, and display two-dimensional shapes, pictures, and text. Because it is a legacy technology, QuickDraw should not be used for any active development. Instead, you should use Quartz.

If your code currently uses QuickDraw, you should begin converting it to Quartz 2D as soon as possible. The QuickDraw API includes features to make transitioning your code easier. For example, QuickDraw includes interfaces for getting a Quartz graphics context from a `GrafPort` structure. You can use these interfaces to transition your QuickDraw code in stages without radically impacting the stability of your builds.

Important: QuickDraw is deprecated in Mac OS X v10.5 and later. QuickDraw is not available for 64-bit applications.

Text and Fonts

Mac OS X provides extensive support for advanced typography for both Carbon and Cocoa programs. These APIs let you control the fonts, layout, typesetting, text input, and text storage in your programs and are described in the following sections. For guidance on choosing the best technology for your needs, see *Getting Started with Text and Fonts*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Cocoa Text

Cocoa provides advanced text-handling capabilities in the Application Kit framework. Based on Core Text, the Cocoa text system provides a multilayered approach to implementing a full-featured text system using Objective-C. This layered approach lets you customize portions of the system that are relevant to your needs while using the default behavior for the rest of the system. You can use Cocoa Text to display small or large amounts of text and can customize the default layout manager classes to support custom layout.

Although part of Cocoa, the Cocoa text system can also be used in Carbon-based applications. If your Carbon application displays moderate amounts of read-only or editable text, you can use `HIView` wrappers for the `NSString`, `NSTextField`, and `NSTextView` classes to implement that support. Using wrappers is much easier than trying to implement the same behavior using lower-level APIs, such as Core Text, ATSUI, or MLTE. For more information on using wrapper classes, see *Carbon-Cocoa Integration Guide*.

For an overview of the Cocoa text system, see *Text System Overview*.

Core Text

Introduced in Mac OS X v10.5, Core Text is a C-based API that provides you with precise control over text layout and typography. Core Text provides a layered approach to laying out and displaying Unicode text. You can modify as much or as little of the system as is required to suit your needs. Core Text also provides optimized configurations for common scenarios, saving setup time in your application. Designed for performance, Core Text is up to twice as fast as ATSUI (see "Apple Type Services for Unicode Imaging" (page 48)), the text-handling technology that it replaces.

The Core Text font API is complementary to the Core Text layout engine. Core Text font technology is designed to handle Unicode fonts natively and comprehensively, unifying disparate Mac OS X font facilities so that developers can do everything they need to do without resorting to other APIs.

Carbon and Cocoa developers who want a high-level text layout API should consider using the Cocoa text system and the supporting Cocoa text views. Unless you need low-level access to the layout manager routines, the Cocoa text system should provide most of the features and performance you need. If you need a lower-level API for drawing any kind of text into a `CGContext`, then you should consider using the Core Text API.

For more information about Core Text, see *Core Text Programming Guide* and *Core Text Reference Collection*.

Apple Type Services

Apple Type Services (ATS) is an engine for the systemwide management, layout, and rendering of fonts. With ATS, users can have a single set of fonts distributed over different parts of the file system or even over a network. ATS makes the same set of fonts available to all clients. The centralization of font rendering and layout contributes to overall system performance by consolidating expensive operations such as synthesizing font data and rendering glyphs. ATS provides support for a wide variety of font formats, including TrueType, PostScript Type 1, and PostScript OpenType. For more information about ATS, see *Apple Type Services for Fonts Programming Guide*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Note: In Mac OS X v10.5 and later, you should consider using the Core Text font-handling API instead of this technology. For more information, see "Core Text" (page 47).

Apple Type Services for Unicode Imaging

Apple Type Services for Unicode Imaging (ATSUI) is the technology behind all text drawing in Mac OS X. ATSUI gives developers precise control over text layout features and supports high-end typography. It is intended for developers of desktop publishing applications or any application that requires the precise manipulation of text. For information about ATSUI, see *ATSUI Programming Guide*.

Note: In Mac OS X v10.5 and later, you should consider using the Core Text API instead of this technology. For more information, see "Core Text" (page 47).

Multilingual Text Engine

The Multilingual Text Engine (MLTE) is an API that provides Carbon-compliant Unicode text editing. MLTE replaces TextEdit and provides an enhanced set of features, including document-wide tabs, text justification, built-in scroll bar handing, built-in printing support, inline input, multiple levels of undo, support for more than 32 KB of text, and support for Apple Type Services. This API is designed for developers who want to incorporate a full set of text editing features into their applications but do not want to worry about managing the text layout or typesetting. For more information about MLTE, see *Handling Unicode Text Editing With MLTE*.

In Mac OS X v10.5 and later, the QuickDraw-related features of MLTE are deprecated. The features that use HITextView are still supported, however.

Note: In Mac OS X v10.5 and later, you should consider using the Core Text API instead of this technology. For more information, see "Core Text" (page 47).

Audio Technologies

Mac OS X includes support for high-quality audio creation and reproduction.

Core Audio

The Core Audio frameworks of Mac OS X offer a sophisticated set of services for manipulating multichannel audio. You can use Core Audio to generate, record, mix, edit, process, and play audio. You can also use Core Audio to generate, record, process, and play MIDI data using both hardware and software MIDI instruments.

For the most part, the interfaces of the Core Audio frameworks are C-based, although some of the Cocoa-related interfaces are implemented in Objective-C. The use of C-based interfaces results in a low-latency, flexible programming environment that you can use from both Carbon and Cocoa applications. Some of the benefits of Core Audio include the following:

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

- Built-in support for reading and writing a wide variety of audio file and data formats
- Plug-in interfaces for handling custom file and data formats
- Plug-in interfaces for performing audio synthesis and audio digital signal processing (DSP)
- A modular approach for constructing audio signal chains
- Scalable multichannel input and output
- Easy synchronization of audio MIDI data during recording or playback
- Support for playing and recording digital audio, including support for scheduled playback and synchronization and for getting timing and control information
- A standardized interface to all built-in and external hardware devices, regardless of connection type (USB, Firewire, PCI, and so on)

For an overview of Core Audio and its features, see *Core Audio Overview*. For reference information, see *Core Audio Framework Reference*.

OpenAL

Introduced in Mac OS X v10.4, the Open Audio Library (OpenAL) audio system adds another way to create audio for your software. The OpenAL interface is a cross-platform standard for delivering 3D audio in applications. It lets you implement high-performance positional audio in games and other programs that require high-quality audio output. Because it is a cross-platform standard, the applications you write using OpenAL on Mac OS X can be ported to run on many other platforms.

In Mac OS X v10.5, several features were incorporated into the existing OpenAL framework. Among these features are support for audio capture, exponential and linear distance models, location offsets, and spatial effects such as reverb and occlusion. In addition, more control is provided for some Core Audio features such as mixer sample rates.

Apple's implementation of OpenAL is based on Core Audio, so it delivers high-quality sound and performance on all Mac OS X systems. To use OpenAL in a Mac OS X application, include the OpenAL framework (`OpenAL.framework`) in your Xcode project. This framework includes header files whose contents conform to the OpenAL specification, which is described at http://www.openal.org.

For more information on the Mac OS X implementation of OpenAL, go to http://developer.apple.com/audio/openal.html.

Video Technologies

The video technologies in Mac OS X allow you to work with movies and other time-based content, including audio.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

QuickTime

QuickTime is a powerful multimedia technology for manipulating, enhancing, and storing video, sound, animation, graphics, text, music, and even 360-degree virtual reality content. It allows you to stream digital video, where the data stream can be either live or stored. QuickTime is a cross-platform technology, supporting Mac OS X, Mac OS 9, Windows 98, Windows Me, Windows 2000, Windows XP, and Windows Vista. Using QuickTime, developers can perform actions such as the following:

- Open and play movie files
- Open and play audio files
- Display still images
- Translate still images from one format to another
- Compress audio, video, and still images
- Synchronize multiple media to a common timeline
- Capture audio and video from an external device
- Stream audio and video over a LAN or the Internet
- Create and display virtual reality objects and panoramas

For a long time, QuickTime has included programming interfaces for the C and C++ languages. Beginning with Mac OS X v10.4, the QuickTime Kit provides an Objective-C based set of classes for managing QuickTime content. For more information about QuickTime Kit, see "QuickTime Kit" (page 51).

Note: In Mac OS X v10.5 and later, you must use the QuickTime Kit framework to create 64-bit applications. The QuickTime C-based APIs are not supported in 64-bit applications.

Supported Media Formats

QuickTime supports more than a hundred media types, covering a range of audio, video, image, and streaming formats. Table 3-2 lists some of the more common file formats it supports. For a complete list of supported formats, see the QuickTime product specification page at http://www.apple.com/quicktime/pro/specs.html.

Table 3-2 Partial list of formats supported by QuickTime

Image formats	PICT, BMP, GIF, JPEG, TIFF, PNG
Audio formats	AAC, AIFF, MP3, WAVE, uLaw
Video formats	AVI, AVR, DV, M-JPEG, MPEG-1, MPEG-2, MPEG-4, AAC, OpenDML, 3GPP, 3GPP2, AMC, H.264
Web streaming formats	HTTP, RTP, RTSP

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Extending QuickTime

The QuickTime architecture is very modular. QuickTime includes media handler components for different audio and video formats. Components also exist to support text display, Flash media, and codecs for different media types. However, most applications do not need to know about specific components. When an application tries to open and play a specific media file, QuickTime automatically loads and unloads the needed components. Of course, applications can specify components explicitly for many operations.

You can extend QuickTime by writing your own component. You might write your own QuickTime component to support a new media type or to implement a new codec. You might also write components to support a custom video capture card. By implementing your code as a QuickTime component that you enable, other applications take advantage of your code and use it to support your hardware or media file formats. See "QuickTime Components" (page 85) for more information.

QuickTime Kit

Introduced in Mac OS X version 10.4, the QuickTime Kit (`QTKit.framework`), is an Objective-C framework for manipulating QuickTime-based media. This framework lets you incorporate movie playback, movie editing, export to standard media formats, and other QuickTime behaviors easily into your applications. The classes in this framework open up a tremendous amount of QuickTime behavior to both Carbon and Cocoa developers. Instead of learning how to use the more than 2500 functions in QuickTime, you can now use a handful of classes to implement the features you need.

In Mac OS X v10.5, support was added for capturing professional-quality audio and video content from one or more external sources, including cameras, microphones, USB and Firewire devices, DV media devices, QuickTime streams, data files, and the screen. The input and output classes included with the framework provide all of the components necessary to implement the most common use case for a media capture system: recording from a camera to a QuickTime file. Video capture includes frame accurate audio/video synchronization, plus you can preview captured content and save it to a file or stream.

Note: The QuickTime Kit framework supersedes the `NSMovie` and `NSMovieView` classes available in Cocoa. If your code uses these older classes, you should change your code to use the QuickTime Kit instead.

For information on how to use the QuickTime Kit, see *QuickTime Kit Programming Guide* and *QTKit Capture Programming Guide*. For reference information about the QuickTime Kit classes, see *QTKit Framework Reference*.

Core Video

Introduced in Mac OS X version 10.4, Core Video provides a modern foundation for delivering video in your applications. It creates a bridge between QuickTime and the graphics card's GPU to deliver hardware-accelerated video processing. By offloading complex processing to the GPU, you can significantly increase performance and reduce the CPU load of your applications. Core Video also allows developers to apply all the benefits of Core Image to video, including filters and effects, per-pixel accuracy, and hardware scalability.

In Mac OS X v10.4, Core Video is part of the Quartz Core framework (`QuartzCore.framework`). In Mac OS X v10.5 and later, the interfaces are duplicated in the Core Video framework (`CoreVideo.framework`).

For information about using the Core Video framework, see *Core Video Programming Guide*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

DVD Playback

Mac OS X version 10.3 and later includes the DVD Playback framework for embedding DVD viewer capabilities into an application. In addition to playing DVDs, you can use the framework to control various aspects of playback, including menu navigation, viewer location, angle selection, and audio track selection. You can play back DVD data from disc or from a local `VIDEO_TS` directory.

For more information about using the DVD Playback framework, see*DVD Playback Services Programming Guide*.

Color Management

ColorSync is the color management system for Mac OS X. It provides essential services for fast, consistent, and accurate color calibration, proofing, and reproduction as well as an interface for accessing and managing systemwide color management settings. It also supports color calibration with hardware devices such as printers, scanners, and displays.

Beginning with Mac OS X version 10.3, the system provides improved support for ColorSync. In most cases, you do not need to call ColorSync functions at all. Quartz and Cocoa automatically use ColorSync to manage pixel data when drawing on the screen. They also respect ICC (International Color Consortium) profiles and apply the system's monitor profile as the source color space. However, you might need to use ColorSync directly if you define a custom color management module (CMM), which is a component that implements color-matching, color-conversion, and gamut-checking services.

For information about the ColorSync API, see *ColorSync Manager Reference*.

Printing

Printing support in Mac OS X is implemented through a collection of APIs and system services available to all application environments. Drawing on the capabilities of Quartz, the printing system delivers a consistent human interface and makes shorter development cycles possible for printer vendors. It also provides applications with a high degree of control over the user interface elements in printing dialogs. Table 3-3 describes some other features of the Mac OS X printing system.

Table 3-3 Features of the Mac OS X printing system

Feature	Description
CUPS	The Common Unix Printing System (CUPS) provides the underlying support for printing. It is an open-source architecture used commonly by the UNIX community to handle print spooling and other low-level features.
Desktop printers	In Mac OS X v10.3 and later, the system supports desktop printers, which offer users a way to manage printing from the Dock or desktop. Users can print natively supported files (like PostScript and PDF) by dragging them to a desktop printer. Users can also manage print jobs.
Fax support	In Mac OS X v10.3 and later, users can fax documents directly from the Print dialog.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Feature	Description
GIMP-Print drivers	In Mac OS X v10.3 and later, the system includes drivers for many older printers through the print facility of the GNU Image Manipulation Program (GIMP).
Native PDF	Supports PDF as a native data type. Any application (except for Classic applications) can easily save textual and graphical data to device-independent PDF where appropriate. The printing system provides this capability from a standard printing dialog.
PostScript support	Mac OS X prints to PostScript Level 2–compatible and Level 3–compatible printers. In Mac OS X v10.3 and later, support is also provided to convert PostScript files directly to PDF.
Print preview	Provides a print preview capability in all environments, except in Classic. The printing system implements this feature by launching a PDF viewer application. This preview is color-managed by ColorSync.
Printer discovery	Printers implementing Bluetooth or Bonjour can be detected, configured, and added to printer lists automatically.
Raster printers	Supports printing to raster printers in all environments, except in the Classic environment.
Speedy spooling	In Mac OS X v10.3 and later, applications that use PDF can submit PDF files directly to the printing system instead of spooling individual pages. This simplifies printing for applications that already store data as PDF.

For an overview of the printing architecture and how to support it, see *Mac OS X Printing System Overview*.

Accelerating Your Multimedia Operations

Mac OS X takes advantage of hardware wherever it can to improve performance wherever it can. In the case of repetitive tasks operating on large data sets, Mac OS X uses the vector-oriented extensions provided by the processor. (Mac OS X currently supports the PowerPC AltiVec extensions and the Intel x86 SSE extensions.) Hardware-based vector units boost the performance of any application that exploits data parallelism, such as those that perform 3D graphic imaging, image processing, video processing, audio compression, and software-based cell telephony. Quartz and QuickTime incorporate vector capabilities, thus any application using these APIs can tap into this hardware acceleration without making any changes.

In Mac OS X v10.3 and later, you can use the Accelerate framework (`Accelerate.framework`) to accelerate complex operations using the available vector unit. This framework supports both the PowerPC AltiVec and Intel x86 SSE extensions internally but provides a single interface for you to use in your application. The advantage of using this framework is that you can simply write your code once without having to code different execution paths for each hardware platform. The functions of this framework are highly tuned for the specific platforms supported by Mac OS X and in many cases can offer better performance than hand-rolled code.

The Accelerate framework is an umbrella framework that wraps the vecLib and vImage frameworks into a single package. The vecLib framework contains vector-optimized routines for doing digital signal processing, linear algebra, and other computationally expensive mathematical operations. (The vecLib framework is also

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

a top-level framework for applications running on versions of Mac OS X up to and including version 10.5.) The vImage framework supports the visual realm, adding routines for morphing, alpha-channel processing, and other image-buffer manipulations.

For information on how to use the components of the Accelerate framework, see *vImage Programming Guide*, *vImage Reference Collection*, and *vecLib Framework Reference*. For general performance-related information, see Reference Library > Performance.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Application Technologies

This chapter summarizes the application-level technologies that are most relevant to developers—that is, that have programmatic interfaces or have an impact on how you write software. It does not describe user-level technologies, such as Exposé, unless there is some aspect of the technology that allows developer involvement.

Application Environments

Applications are by far the predominant type of software created for Mac OS X, or for any platform. Mac OS X provides numerous environments for developing applications, each of which is suited for specific types of development. The following sections describe each of the primary application environments and offer guidelines to help you choose an environment that is appropriate for your product requirements.

Important: With the transition to Intel-based processors, developers should always create universal binaries for their Carbon, Cocoa, and BSD applications. Java and WebObjects may also need to create universal binaries for bridged code. For information on how to create universal binaries, see *Universal Binary Programming Guidelines, Second Edition*.

Cocoa

Cocoa is an object-oriented environment designed for rapid application development. It features a sophisticated framework of objects for implementing your application and takes full advantage of graphical tools such as Interface Builder to enable you to create full-featured applications quickly and without a lot of code. The Cocoa environment is especially suited for:

- New developers
- Developers who prefer working with object-oriented systems
- Developers who need to prototype an application quickly
- Developers who prefer to leverage the default behavior provided by the Cocoa frameworks so they can focus on the features unique to their application
- Objective-C or Objective-C++ developers
- Python and Ruby developers who want to take advantage of Cocoa features; see *Ruby and Python Programming Topics for Mac OS X*

The objects in the Cocoa framework handle much of the behavior required of a well-behaved Mac OS X application, including menu management, window management, document management, Open and Save dialogs, and pasteboard (clipboard) behavior. Cocoa's support for Interface Builder means that you can create most of your user interface (including much of its behavior) graphically rather than programatically. With the addition of Cocoa bindings and Core Data, you can also implement most of the rest of your application graphically as well.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

The Cocoa application environment consists of two object-oriented frameworks: Foundation (`Foundation.framework`) and the Application Kit (`AppKit.framework`). The classes in the Foundation framework implement data management, file access, process notification, memory management, network communication, and other low-level features. The classes in the Application Kit framework implement the user interface layer of an application, including windows, dialogs, controls, menus, and event handling. If you are writing an application, link with the Cocoa framework (`Cocoa.framework`), which imports both the Foundation and Application Kit frameworks. If you are writing a Cocoa program that does not have a graphical user interface (a background server, for example), you can link your program solely with the Foundation framework.

Apple's developer documentation contains a section devoted to Cocoa where you can find conceptual material, reference documentation, and tutorials showing how to write Cocoa applications. If you are a new Cocoa developer, be sure to read *Cocoa Fundamentals Guide*, which provides an in-depth overview of the development process for Cocoa applications. For information about the development tools, including Interface Builder, see "Mac OS X Developer Tools" (page 127).

Carbon

Based on the original Mac OS 9 interfaces, the Carbon application environment is a set of C APIs used to create full-featured applications for all types of users. The Carbon environment includes support for all of the standard Aqua user interface elements such as windows, controls, and menus. It also provides an extensive infrastructure for handling events, managing data, and using system resources.

The Carbon environment is especially suited for:

- Mac OS 9 developers porting their applications to Mac OS X
- Developers who prefer to work solely in C or C++
- Developers who are porting commercial applications from other procedural-based systems and want to use as much of their original code as possible

Because the Carbon interfaces are written in C, some developers may find them more familiar than the interfaces in the Cocoa or Java environments. Some C++ developers may also prefer the Carbon environment for development, although C++ code can be integrated seamlessly into Cocoa applications as well.

The Carbon APIs offer you complete control over the features in your application; however, that control comes at the cost of added complexity. Whereas Cocoa provides many features for you automatically, with Carbon you must write the code to support those features yourself. For example, Cocoa applications automatically implement support for default event handlers, the pasteboard, and Apple events, but Carbon developers must add support for these features themselves.

In Mac OS X v10.5 and later, Carbon includes support for integrating Cocoa views into your Carbon applications. After creating the Cocoa view, you can wrap it in an HIView object and embed that object in your window. Once embedded, you use the standard HIView functions to manipulate the view. Wrapped Cocoa views can be used in both composited and noncomposited windows to support views and controls that are available in Cocoa but are not yet available in Carbon. For more information, see *Carbon-Cocoa Integration Guide* and *HIView Reference*.

The Carbon application environment comprises several key umbrella frameworks, including the Carbon framework (`Carbon.framework`), the Core Services framework (`CoreServices.framework`), and the Application Services framework (`ApplicationServices.framework`). The Carbon environment also uses the Core Foundation framework (`CoreFoundation.framework`) extensively in its implementation.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Apple's developer documentation contains a section devoted to Carbon, where you can find conceptual material, reference documentation, and tutorials showing how to write applications using Carbon. See *Getting Started with Carbon* in Carbon Documentation for an overview of the available Carbon documentation.

If you are migrating a Mac OS 9 application to Mac OS X, read *Carbon Porting Guide*. If you are migrating from Windows, see *Porting to Mac OS X from Windows Win32 API*. If you are migrating from UNIX, see *Porting UNIX/Linux Applications to Mac OS X*.

Java

The Java application environment is a runtime environment and set of objects for creating applications that run on multiple platforms. The Java environment is especially suited for:

- Experienced Java Platform, Standard Edition/Java SE developers
- Developers writing applications to run on multiple platforms
- Developers writing Java applets for inclusion in web-based content
- Developers familiar with the Swing or AWT toolkits for creating graphical interfaces

The Java application environment lets you develop and execute 100% pure Java applications and applets. This environment conforms to the specifications laid out by the J2SE platform, including those for the Java virtual machine (JVM), making applications created with this environment very portable. You can run them on computers with a different operating system and hardware as long as that system is running a compatible version of the JVM. Java applets should run in any Internet browser that supports them.

Note: Any Mach-O binaries that interact with the JVM must be universal binaries. This includes JNI libraries as well as traditional applications that invoke the JVM. For more information, see *Universal Binary Programming Guidelines, Second Edition*.

For details on the tools and support provided for Java developers, see "Java Support" (page 38).

AppleScript

The AppleScript application environment lets you use AppleScript scripts to quickly create native Mac OS X applications that support the Aqua user interface guidelines. At the heart of this environment is AppleScript Studio, which combines features from AppleScript with Xcode, Interface Builder, and the Cocoa application framework. Using these tools, you can create applications that use AppleScript scripts to control a broad range of Cocoa user-interface objects.

AppleScript Studio has something to offer both to scripters and to those with Cocoa development experience. In addition to AppleScript's ability to control multiple applications, including parts of the Mac OS itself, you can use it for the following:

- Scripters can now create applications with windows, buttons, menus, text fields, tables, and much more. Scripts have full access to user interface objects.
- Cocoa developers can take advantage of AppleScript Studio's enhanced Cocoa scripting support, which can be useful in prototyping, testing, and deploying applications.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

For information on how to create applications using AppleScript Studio, see *AppleScript Studio Programming Guide*.

WebObjects

The WebObjects application environment is a set of tools and object-oriented frameworks targeted at developers creating web services and web-based applications. The WebObjects environment provides a set of flexible tools for creating full-featured web applications. Common uses for this environment include the following:

- Creating a web-based interface for dynamic content, including programmatically generated content or content from a database
- Creating web services based on SOAP, XML, and WSDL

WebObjects is a separate product sold by Apple. If you are thinking about creating a web storefront or other web-based services, see the information available at http://developer.apple.com/tools/webobjects.

> **Note:** If your WebObjects application includes bridged code in a Mach-O binary, you need to create a universal binary for the Mach-O binary code. For more information, see *Universal Binary Programming Guidelines, Second Edition*.

BSD and X11

The BSD application environment is a set of low-level interfaces for creating shell scripts, command-line tools, and daemons. The BSD environment is especially suited for:

- UNIX developers familiar with the FreeBSD and POSIX interfaces
- Developers who want to create text-based scripts and tools, rather than tools that have a graphical user interface
- Developers who want to provide fundamental system services through the use of daemons or other root processes

The BSD environment is for developers who need to work below the user interface layers provided by Carbon, Cocoa, and WebObjects. Developers can also use this environment to write command-line tools or scripts to perform specific user-level tasks.

X11 extends the BSD environment by adding a set of programming interfaces for creating graphical applications that can run on a variety of UNIX implementations. The X11 environment is especially suited for developers who want to create graphical applications that are also portable across different varieties of UNIX.

The BSD environment is part of the Darwin layer of Mac OS X. For information about Darwin, see Reference Library > Darwin. For more information about X11 development, see http://developer.apple.com/darwin/projects/X11. See also "Information on BSD" (page 15) for links to additional BSD resources.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Application Technologies

Mac OS X includes several technologies that make developing applications easier. These technologies range from utilities for managing your internal data structures to high-level frameworks for burning CDs and DVDs. This section summarizes the application-level technologies that are relevant to developers—that is, that have programmatic interfaces or have an impact on how you write software. It does not describe user-level technologies, such as Exposé, unless there is some aspect of the technology that allows developer involvement.

If you are new to developing Mac OS X software, you should read through this chapter at least once to understand the available technologies and how you might use them in your software. Even experienced developers should revisit this chapter periodically to remind themselves of the available technologies.

Address Book Framework

Introduced in Mac OS X v10.2, Address Book is technology that encompasses a centralized database for contact and group information, an application for viewing that information, and a programmatic interface for accessing that information in your own programs. The database contains information such as user names, street addresses, email addresses, phone numbers, and distribution lists. Applications that support this type of information can use this data as is or extend it to include application-specific information.

The Address Book framework (`AddressBook.framework`) provides your application with a way to access user records and create new ones. Applications that support this framework gain the ability to share user records with other applications, such as the Address Book application and the Apple Mail program. The framework also supports the concept of a "Me" record, which contains information about the currently logged-in user. You can use this record to provide information about the current user automatically; for example, a web browser might use it to populate a web form with the user's address and phone number.

For more information about this technology, see *Address Book Programming Guide for Mac OS X* and either *Address Book Objective-C Framework Reference* or *Address Book C Framework Reference*.

Automator Framework

Introduced in Mac OS X v10.5, the Automator framework (`Automator.framework`) adds support for running workflows from your applications. Workflows are products of the Automator application; they string together the actions defined by various applications to perform complex tasks automatically. Unlike AppleScript, which uses a scripting language to implement the same behavior, workflows are constructed visually, requiring no coding or scripting skills to create.

For information about incorporating workflows into your own applications, see *Automator Framework Reference*.

Bonjour

Introduced in Mac OS X version 10.2, Bonjour is Apple's implementation of the zero-configuration networking architecture, a powerful system for publishing and discovering services over an IP network. It is relevant to both software and hardware developers.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Incorporating Bonjour support into your software offers a significant improvement to the overall user experience. Rather than prompt the user for the exact name and address of a network device, you can use Bonjour to obtain a list of available devices and let the user choose from that list. For example, you could use it to look for available printing services, which would include any printers or software-based print services, such as a service to create PDF files from print jobs.

Developers of network-based hardware devices are strongly encouraged to support Bonjour. Bonjour alleviates the need for complicated setup instructions for network-based devices such as printers, scanners, RAID servers, and wireless routers. When plugged in, these devices automatically publish the services they offer to clients on the network.

For information on how to incorporate Bonjour services into a Cocoa application, see *Bonjour Overview*. Bonjour for non-Cocoa applications is described in *DNS Service Discovery Programming Guide*.

Calendar Store Framework

Introduced in Mac OS X v10.5, the Calendar Store framework (`CalendarStore.framework`) lets you access iCal data from an Objective-C based application. You can use this framework to fetch user calendars, events, and tasks from the iCal data storage, receive notifications when those objects change, and make changes to the user's calendar.

For information about using the Calendar Store framework, see [Calendar Store Programming Guide] and *Calendar Store Programming Guide*.

Core Data Framework

Introduced in Mac OS X version 10.4, the Core Data framework (`CoreData.framework`) manages the data model of a Cocoa-based Model-View-Controller application. Core Data is intended for use in applications where the data model is already highly structured. Instead of defining data structures programmatically, you use the graphical tools in Xcode to build a schema representing your data model. At runtime, instances of your data-model entities are created, managed, and made available through the Core Data framework with little or no coding on your part.

By managing your application's data model for you, Core Data significantly reduces the amount of code you have to write for your application. Core Data also provides the following features:

- Storage of object data in mediums ranging from an XML file to a SQLite database
- Management of undo/redo beyond basic text editing
- Support for validation of property values
- Support for propagating changes and ensuring that the relationships between objects remain consistent
- Grouping, filtering, and organizing data in memory and transferring those changes to the user interface through Cocoa bindings

If you are starting to develop a new application, or are planning a significant update to an existing application, you should consider using Core Data. For more information about Core Data, including how to use it in your applications, see *Core Data Programming Guide*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Disc Recording Framework

Introduced in Mac OS X version 10.2, the Disc Recording framework (`DiscRecording.framework`) gives applications the ability to burn and erase CDs and DVDs. This framework was built to satisfy the simple needs of a general application, making it easy to add basic audio and data burning capabilities. At the same time, the framework is flexible enough to support professional CD and DVD mastering applications.

The Disc Recording framework minimizes the amount of work your application must perform to burn optical media. Your application is responsible for specifying the content to be burned but the framework takes over the process of buffering the data, generating the proper file format information, and communicating everything to the burner. In addition, the Disc Recording UI framework (`DiscRecordingUI.framework`) provides a complete, standard set of windows for gathering information from the user and displaying the progress of the burn operation.

The Disc Recording framework supports applications built using Carbon and Cocoa. The Disc Recording UI framework currently provides user interface elements for Cocoa applications only.

For more information, see *Disc Recording Framework Reference* and *Disc Recording UI Framework Reference*.

Help Support

Although some applications are extremely simple to use, most require some documentation. Help tags (also called tooltips) and documentation are the best ways to provide users with immediate answers to questions. Help tags provide descriptive information about your user interface quickly and unobtrusively. Documentation provides more detailed solutions to problems, including conceptual material and task-based examples. Both of these elements help the user understand your user interface better and should be a part of every application.

In Mac OS X v10.5 and later, the Spotlight for Help feature makes it easier for users to locate items in complex menu bars. For applications with a Help menu, Mac OS X automatically inserts a special search field menu item at the top of the menu. When the user enters a string in this search field, the system searches the application menus for commands containing the specified string. Moving the mouse over a search result reveals the location of that item in the menus. Developers do not need to add any code to their applications to support this feature.

For information on adding help to a Cocoa application, see *Online Help*. For information on adding help to a Carbon application, see *Apple Help Programming Guide*.

Human Interface Toolbox

Introduced in Mac OS X version 10.2, the Human Interface Toolbox (HIToolbox) provides a modern set of interfaces for creating and managing windows, controls, and menus in Carbon applications. The HIObject model builds on Core Foundation data types to bring a modern, object-oriented approach to the HIToolbox. Although the model is object-oriented, access to the objects is handled by a set of C interfaces. Using the HIToolbox interfaces is recommended for the development of new Carbon applications. Some benefits of this technology include the following:

- Drawing is handled natively using Quartz.
- A simplified, modern coordinate system is used that is not bounded by the 16-bit space of QuickDraw.
- Support for arbitrary views is provided.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

- Layering of views is handled automatically.
- Views can be attached and detached from windows.
- Views can be hidden temporarily.
- You can use Interface Builder to create your interfaces.

Note: The HIToolbox interfaces are available for creating 32-bit applications only. If you are creating 64-bit applications, you should use Cocoa for your user interface instead.

For reference material and an overview of HIObject and other HIToolbox objects, see the documents in Reference Library > Carbon > Human Interface Toolbox.

Identity Services

Introduced in Mac OS X v10.5, Identity Services encompasses features located in the Collaboration and Core Services frameworks. Identity Services provides a way to manage groups of users on a local system. In addition to standard login accounts, administrative users can now create **sharing** accounts, which use access control lists to restrict access to designated system or application resources. Sharing accounts do not have an associated home directory on the system and have much more limited privileges than traditional login accounts.

The Collaboration framework (`Collaboration.framework`) provides a set of Objective-C classes for displaying sharing account information and other identity-related user interfaces. The classes themselves are wrappers for the C-based identity management routines found in the Core Services framework. Applications can use either the Objective-C or C-based APIs to display information about users and groups and display a panel for selecting users and groups during the editing of access control lists.

For more information about the features of Identity Services and how you use those features in your applications, see *Identity Services Programming Guide* and *Identity Services Reference Collection*.

Instant Message Framework

Introduced in Mac OS X version 10.4, the Instant Message framework (`InstantMessage.framework`) supports the detection and display of a user's online presence in applications other than iChat. You can find out the current status of a user connected to an instant messaging service, obtain the user's custom icon and status message, or obtain a URL to a custom image that indicates the user's status. You can use this information to display the user's status in your own application. For example, Mail identifies users who are currently online by tagging that user's email address with a special icon.

In Mac OS X v10.5, you can use the Instant Message framework to support iChat Theater. This feature gives your application the ability to inject audio or video content into a running iChat conference. The content you provide is then mixed with the user's live microphone and encoded automatically into the H.264 video format for distribution to conference attendees.

For more information about using the Instant Message framework, see *Instant Message Programming Guide*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Image Capture Services

The Image Capture Services framework (part of `Carbon.framework`) is a high-level framework for capturing image data from scanners and digital cameras. The interfaces of the framework are device-independent, so you can use it to gather data from any devices connected to the system. You can get a list of devices, retrieve information about a specific device or image, and retrieve the image data itself.

This framework works in conjunction with the Image Capture Devices framework (`ICADevices.framework`) to communicate with imaging hardware. For information on how to use the Image Capture Services framework, see *Image Capture Applications Programming Guide*.

Ink Services

The Ink feature of Mac OS X provides handwriting recognition for applications that support the Carbon and Cocoa text systems (although the automatic support provided by these text systems is limited to basic recognition). The Ink framework offers several features that you can incorporate into your applications, including the following:

- Enable or disable handwriting recognition programmatically.
- Access Ink data directly.
- Support either deferred recognition or recognition on demand.
- Support the direct manipulation of text by means of gestures.

The Ink framework is not limited to developers of end-user applications. Hardware developers can also use it to implement a handwriting recognition solution for a new input device. You might also use the Ink framework to implement your own correction model to provide users with a list of alternate interpretations for handwriting data.

Ink is included as a subframework of `Carbon.framework`. For more information on using Ink in Carbon and Cocoa applications, see *Using Ink Services in Your Application*.

Input Method Kit Framework

Introduced in Mac OS X v10.5, the Input Method Kit (`InputMethodKit.framework`) is an Objective-C framework for building input methods for Chinese, Japanese, and other languages. The Input Method Kit framework lets developers focus exclusively on the development of their input method product's core behavior: the text conversion engine. The framework handles tasks such as connecting to clients, running candidate windows, and several other common tasks that developers would normally have to implement themselves.

For information about its classes, see *Input Method Kit Framework Reference*.

Keychain Services

Keychain Services provides a secure way to store passwords, keys, certificates, and other sensitive information associated with a user. Users often have to manage multiple user IDs and passwords to access various login accounts, servers, secure websites, instant messaging services, and so on. A keychain is an encrypted container

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

that holds passwords for multiple applications and secure services. Access to the keychain is provided through a single master password. Once the keychain is unlocked, Keychain Services–aware applications can access authorized information without bothering the user.

Users with multiple accounts tend to manage those accounts in the following ways:

- They create a simple, easily remembered password.
- They repeatedly use the same password.
- They write the password down where it can easily be found.

If your application handles passwords or sensitive information, you should add support for Keychain Services into your application. For more information on this technology, see *Keychain Services Programming Guide*.

Latent Semantic Mapping Services

Introduced in Mac OS X v10.5, the Latent Semantic Mapping framework (`LatentSemanticMapping.framework`) contains a Unicode-based API that supports the classification of text and other token-based content into developer-defined categories, based on semantic information latent in the text. Using this API and text samples with known characteristics, you create and train maps, which you can use to analyze and classify arbitrary text. You might use such a map to determine, for example, if an email message is consistent with the user's interests.

For information about the Latent Semantic Mapping framework, see *Latent Semantic Mapping Reference*.

Launch Services

Launch Services provides a programmatic way for you to open applications, documents, URLs, or files with a given MIME type in a way similar to the Finder or the Dock. It makes it easy to open documents in the user's preferred application or open URLs in the user's favorite web browser. The Launch Services framework also provides interfaces for programmatically registering the document types your application supports.

For information on how to use Launch Services, see *Launch Services Programming Guide*.

Open Directory

Open Directory is a directory services architecture that provides a centralized way to retrieve information stored in local or network databases. Directory services typically provide access to collected information about users, groups, computers, printers, and other information that exists in a networked environment (although they can also store information about the local system). You use Open Directory in your programs to retrieve information from these local or network databases. For example, if you're writing an email program, you can use Open Directory to connect to a corporate LDAP server and retrieve the list of individual and group email addresses for the company.

Open Directory uses a plug-in architecture to support a variety of retrieval protocols. Mac OS X provides plug-ins to support LDAPv2, LDAPv3, NetInfo, AppleTalk, SLP, SMB, DNS, Microsoft Active Directory, and Bonjour protocols, among others. You can also write your own plug-ins to support additional protocols.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

For more information on this technology, see *Open Directory Programming Guide*. For information on how to write Open Directory plug-ins, see *Open Directory Plug-in Programming Guide*.

PDF Kit Framework

Introduced in Mac OS X version 10.4, PDF Kit is a Cocoa framework for managing and displaying PDF content directly from your application's windows and dialogs. Using the classes of the PDF Kit, you can embed a PDFView in your window and give it a PDF file to display. The PDFView class handles the rendering of the PDF content, handles copy-and-paste operations, and provides controls for navigating and setting the zoom level. Other classes let you get the number of pages in a PDF file, find text, manage selections, add annotations, and specify the behavior of some graphical elements, among other actions. Users can also copy selected text in a PDFView to the pasteboard.

Note: Although it is written in Objective-C, you can use the classes of the PDF Kit in both Carbon and Cocoa applications. For information on how to do this, see *Carbon-Cocoa Integration Guide*.

If you need to display PDF data directly from your application, the PDF Kit is highly recommended. It hides many of the intricacies of the Adobe PDF specification and provides standard PDF viewing controls automatically. The PDF Kit is part of the Quartz framework (`Quartz.framework`). For more information, see *PDF Kit Programming Guide*.

Publication Subscription Framework

Introduced in Mac OS X v10.5, the Publication Subscription framework (`PubSub.framework`) is a new framework that provides high-level support for subscribing to RSS and Atom feeds. You can use the framework to subscribe to podcasts, photocasts, and any other feed-based document. The framework handles all the feed downloads and updates automatically and provides your application with the data from the feed.

For information about the Publication Subscription framework, see *Publication Subscription Programming Guide* and *Publication Subscription Framework Reference*.

Search Kit Framework

Introduced in Mac OS X version 10.3, the Search Kit framework lets you search, summarize, and retrieve documents written in most human languages. You can incorporate these capabilities into your application to support fast searching of content managed by your application.

The Search Kit framework is part of the Core Services umbrella framework. The technology is derived from the Apple Information Access Toolkit, which is often referred to by its code name V-Twin. Many system applications, including Spotlight, Finder, Address Book, Apple Help, and Mail use this framework to implement searching.

Search Kit is an evolving technology and as such continues to improve in speed and features. For detailed information about the available features, see *Search Kit Reference*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Security Services

Mac OS X security is built using several open source technologies, including BSD, Common Data Security Architecture (CDSA), and Kerberos. Mac OS X builds on these basic technologies by implementing a layer of high-level services to simplify your security solutions. These high-level services provide a convenient abstraction and make it possible for Apple and third parties to implement new security features without breaking your code. They also make it possible for Apple to combine security technologies in unique ways; for example, Keychain Services provides encrypted data storage with authenticated access using several CDSA technologies.

Mac OS X provides high-level interfaces for the following features:

- User authentication
- Certificate, key, and trust services
- Authorization services
- Secure transport
- Keychain Services

Mac OS X supports many network-based security standards, including SFTP, S/MIME, and SSH. For a complete list of network protocols, see "Standard Network Protocols" (page 24).

For more information about the security architecture and security-related technologies of Mac OS X, see *Security Overview*. For additional information about CDSA, see the following page of the Open Group's website: http://www.opengroup.org/security/cdsa.htm.

Speech Technologies

Mac OS X contains speech technologies that recognize and speak U.S. English. These technologies provide benefits for users and present the possibility of a new paradigm for human-computer interaction.

Speech recognition is the ability for the computer to recognize and respond to a person's speech. Using speech recognition, users can accomplish tasks comprising multiple steps with one spoken command. Because users control the computer by voice, speech-recognition technology is very important for people with special needs. You can take advantage of the speech engine and API included with Mac OS X to incorporate speech recognition into your applications.

Speech synthesis, also called text-to-speech (TTS), converts text into audible speech. TTS provides a way to deliver information to users without forcing them to shift attention from their current task. For example, the computer could deliver messages such as "Your download is complete" and "You have email from your boss; would you like to read it now?" in the background while you work. TTS is crucial for users with vision or attention disabilities. As with speech recognition, Mac OS X TTS provides an API and several user interface features to help you incorporate speech synthesis into your applications. You can also use speech synthesis to replace digital audio files of spoken text. Eliminating these files can reduce the overall size of your software bundle.

For more information, see Reference Library > User Experience > Speech Technologies.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

SQLite Library

Introduced in Mac OS X version 10.4, the SQLite library lets you embed a SQL database engine into your applications. Programs that link with the SQLite library can access SQL databases without running a separate RDBMS process. You can create local database files and manage the tables and records in those files. The library is designed for general purpose use but is still optimized to provide fast access to database records.

The SQLite library is located at `/usr/lib/libsqlite3.dylib` and the `sqlite3.h` header file is in `/usr/include`. A command-line interface (`sqlite3`) is also available for communicating with SQLite databases using scripts. For details on how to use this command-line interface, see `sqlite3` man page.

For more information about using SQLite, go to http://www.sqlite.org.

Sync Services Framework

Introduced in Mac OS X version 10.4, the Sync Services framework gives you access to the data synchronization engine built-in to Mac OS X. You can use this framework to synchronize your application data with system databases, such as those provided by Address Book and iCal. You can also publish your application's custom data types and make them available for syncing. You might do this to share your application's data with other applications on the same computer or with applications on multiple computers (through the user's .Mac account).

With the Sync Services framework, applications can directly initiate the synchronization process. Prior to Mac OS X v10.4, synchronization occurred only through the iSync application. In Mac OS X v10.4 and later, the iSync application still exists but is used to initiate the synchronization process for specific hardware devices, like cell phones.

The Sync Services framework (`SyncServices.framework`) provides an Objective-C interface but can be used by both Carbon and Cocoa applications. Applications can use this framework to initiate sync sessions and to push and pull records from the central "truth" database, which the sync engine uses to maintain the master copy of the synchronized records. The system provides predefined schemas for contacts, calendars, bookmarks, and mail notes (see *Apple Applications Schema Reference*). You can also distribute custom schemas for your own data types and register them with Sync Services.

For more information about using Sync Services in your application, see *Sync Services Programming Guide* and *Sync Services Framework Reference*.

Web Kit Framework

Introduced in Mac OS X version 10.3, the Web Kit framework provides an engine for displaying HTML-based content. The Web Kit framework is an umbrella framework containing two subframeworks: Web Core and JavaScript Core. The Web Core framework is based on the kHTML rendering engine, an open source engine for parsing and displaying HTML content. The JavaScript Core framework is based on the KJS open source library for parsing and executing JavaScript code.

Starting with Mac OS X version 10.4, Web Kit also lets you create text views containing editable HTML. The editing support is equivalent to the support available in Cocoa for editing RTF-based content. With this support, you can replace text and manipulate the document text and attributes, including CSS properties.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Although it offers many features, the Web Kit editing support is not intended to provide a full-featured editing facility like you might find in professional HTML editing applications. Instead, it is aimed at developers who need to display HTML and handle the basic editing of HTML content.

Also introduced in Mac OS X version 10.4, Web Kit includes support for creating and editing content at the DOM level of an HTML document. You can use this support to navigate DOM nodes and manipulate those nodes and their attributes. You can also use the framework to extract DOM information. For example, you could extract the list of links on a page, modify them, and replace them prior to displaying the document in a web view.

For information on how to use Web Kit from both Carbon and Cocoa applications, see *WebKit Objective-C Programming Guide*. For information on the classes and protocols in the Web Kit framework, see *WebKit Objective-C Framework Reference*.

Time Machine Support

Introduced in Mac OS X v10.5, the Time Machine feature protects user data from accidental loss by automatically backing up data to a different hard drive. Included with this feature is a set of programmer-level functions that you can use to exclude unimportant files from the backup set. For example, you might use these functions to exclude your application's cache files or any files that can be recreated easily. Excluding these types of files improves backup performance and reduces the amount of space required to back up the user's system.

For information about the new functions, see *Backup Core Reference*.

Web Service Access

Many businesses provide web services for retrieving data from their websites. The available services cover a wide range of information and include things such as financial data and movie listings. Mac OS X has included support for calling web-based services using Apple events since version 10.1. However, starting with version 10.2, the Web Services Core framework (part of the Core Services umbrella framework) provides support for the invocation of web services using CFNetwork.

For a description of web services and information on how to use the Web Services Core framework, see *Web Services Core Programming Guide*.

XML Parsing Libraries

In Mac OS X v10.3, the Darwin layer began including the `libXML2` library for parsing XML data. This is an open source library that you can use to parse or write arbitrary XML data quickly. The headers for this library are located in the `/usr/include/libxml2` directory.

Several other XML parsing technologies are also included in Mac OS X. For arbitrary XML data, Core Foundation provides a set of functions for parsing the XML content from Carbon or other C-based applications. Cocoa provides several classes to implement XML parsing. If you need to read or write a property list file, you can use either the Core Foundation CFPropertyList functions or the Cocoa NSDictionary object to build a set of collection objects with the XML data.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

For information on Core Foundation support for XML parsing, see the documents in Reference Library > Core Foundation > Data Management. For information on parsing XML from a Cocoa application, see *Tree-Based XML Programming Guide for Cocoa*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

User Experience

One reason users choose the Macintosh over other platforms is that it provides a compelling user experience. This user experience is defined partly by the technologies and applications that are built-in to Mac OS X and partly by the applications you create. Your applications play a key role in delivering the experience users expect. This means that your applications need to support the features that help them blend into the Mac OS X ecosystem and create a seamless user experience.

Technologies

The following sections describe technologies that form a key part of the Mac OS X user experience. If you are developing an application, you should consider adopting these technologies to make sure your application integrates cleanly into Mac OS X. Most of these technologies require little effort to support but provide big advantages in your software's usability and in the likelihood of user adoption.

Aqua

Aqua defines the appearance and overall behavior of Mac OS X applications. Aqua applications incorporate color, depth, translucence, and complex textures into a visually appealing interface. The behavior of Aqua applications is consistent, providing users with familiar paradigms and expected responses to user-initiated actions.

Applications written using modern Mac OS X interfaces (such as those provided by Carbon and Cocoa) get much of the Aqua appearance automatically. However, there is more to Aqua than that. Interface designers must still follow the Aqua guidelines to position windows and controls in appropriate places. Designers must take into account features such as text, keyboard, and mouse usage and make sure their designs work appropriately for Aqua. The implementers of an interface must then write code to provide the user with appropriate feedback and to convey what is happening in an informative way.

Apple provides the Interface Builder application to assist developers with the proper layout of interfaces. However, you should also be sure to read *Apple Human Interface Guidelines*, which provides invaluable advice on how to create Aqua-compliant applications and on the best Mac OS X interface technologies to use.

Quick Look

Introduced in Mac OS X v10.5, Quick Look is a technology that enables client applications, such as Spotlight and the Finder, to display thumbnail images and full-size previews of documents. Mac OS X provides automatic support for many common content types, including HTML, RTF, plain text, TIFF, PNG, JPEG, PDF, and QuickTime movies. If your application defines custom document formats, you should provide a Quick Look generator for those formats. Generators are plug-ins that convert documents of the appropriate type from their native format to a format that Quick Look can display to users. Mac OS X makes extensive use of generators to give users quick previews of documents without having to open the corresponding applications.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

For information about supporting Quick Look for your custom document types, see *Quick Look Programming Guide* and *Quick Look Framework Reference*.

Resolution-Independent User Interface

Resolution independence decouples the resolution of the user's screen from the units you use in your code's drawing operations. While Mac OS X version 10.4 and earlier assumed a screen resolution of 72 dots per inch (dpi), most modern screens actually have resolutions that are 100 dpi or more. The result of this difference is that content rendered for a 72 dpi screen appears smaller on such screens—a problem that will only get worse as screen resolutions increase.

In Mac OS X v10.4, steps were taken to support content scaling for screen-based rendering. In particular, the notion of a scale factor was introduced to the system, although not heavily publicized. This scale factor was fixed at 1.0 by default but could be changed by developers using the Quartz Debug application. In addition, Carbon and Cocoa frameworks were updated to support scale factors and interfaces were introduced to return the current screen scale factor so that developers could begin testing their applications in a content-scaled world.

Although the Mac OS X frameworks handle many aspects related to resolution-independent drawing, there are still things you need to do in your drawing code to support resolution independence:

- Update the images and artwork in your user interface. As the pixel density of displays increases, you need to make sure your application's custom artwork can scale accordingly—that is, your art needs to be larger in terms of pixel dimensions to avoid looking pixellated at higher scale factors. This includes changing:
  - Application icons
  - Images that appear in buttons or other controls
  - Other custom images you use in your interface
- Update code that relies on precise pixel alignment to take the current scale factor into account. Both Cocoa and Carbon provide ways to access the current scale factor.
- Consider drawing lines, fills, and gradients programmatically instead of using prerendered images. Shapes drawn using Quartz and Cocoa always scale appropriately to the screen resolution.

When scaling your images, be sure to cache the scaled versions of frequently-used images to increase drawing efficiency. For more information about resolution-independence and how to support it in your code, see *Resolution Independence Guidelines*.

Spotlight

Introduced in Mac OS X version 10.4, Spotlight provides advanced search capabilities for applications. The Spotlight server gathers metadata from documents and other relevant user files and incorporates that metadata into a searchable index. The Finder uses this metadata to provide users with more relevant information about their files. For example, in addition to listing the name of a JPEG file, the Finder can also list its width and height in pixels.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Application developers use Spotlight in two different ways. First, you can search for file attributes and content using the Spotlight search API. Second, if your application defines its own custom file formats, you should incorporate any appropriate metadata information in those formats and provide a Spotlight importer plug-in to return that metadata to Spotlight.

Note: You should not use Spotlight for indexing and searching the general content of a file. Spotlight is intended for searching only the meta information associated with files. To search the actual contents of a file, use the Search Kit API. For more information, see "Search Kit Framework" (page 65).

In Mac OS X v10.5 and later, several new features were added to make working with Spotlight easier. The File manager includes functions for swapping the contents of a file while preserving its original metadata; see the `Files.h` header file in the Core Services framework. Spotlight also defines functions for storing lineage information with a file so that you can track modifications to that file.

For more information on using Spotlight in your applications, see *Spotlight Overview*.

Bundles and Packages

A feature integral to Mac OS X software distribution is the bundle mechanism. Bundles encapsulate related resources in a hierarchical file structure but present those resources to the user as a single entity. Programmatic interfaces make it easy to find resources inside a bundle. These same interfaces form a significant part of the Mac OS X internationalization strategy.

Applications and frameworks are only two examples of bundles in Mac OS X. Plug-ins, screen savers, and preference panes are all implemented using the bundle mechanism as well. Developers can also use bundles for their document types to make it easier to store complex data.

Packages are another technology, similar to bundles, that make distributing software easier. A package—also referred to as an installation package—is a directory that contains files and directories in well-defined locations. The Finder displays packages as files. Double-clicking a package launches the Installer application, which then installs the contents of the package on the user's system.

For an overview of bundles and how they are constructed, see *Bundle Programming Guide*. For information on how to package your software for distribution, see *Software Delivery Guide*.

Code Signing

In Mac OS X v10.5 and later, it is possible to associate a digital signature with your application using the `codesign` command-line tool. If you have a certificate that is authorized for signing, you can use that certificate to sign your application's code file. Signing your application makes it possible for Mac OS X to verify the source of the application and ensure the application has not changed since it was shipped. If the application has been tampered with, Mac OS X detects the change and can alert the user to the problem. Signed applications also make it harder to circumvent parental controls and other protection features of the system.

For information on signing your application, see *Code Signing Guide*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Internationalization and Localization

Localizing your application is necessary for success in many foreign markets. Users in other countries are much more likely to buy your software if the text and graphics reflect their own language and culture. Before you can localize an application, though, you must design it in a way that supports localization, a process called internationalization. Properly internationalizing an application makes it possible for your code to load localized content and display it correctly.

Internationalizing an application involves the following steps:

- Use Unicode strings for storing user-visible text.
- Extract user-visible text into "strings" resource files.
- Use nib files to store window and control layouts whenever possible.
- Use international or culture-neutral icons and graphics whenever possible.
- Use Cocoa or Core Text to handle text layout.
- Support localized file-system names (also known as "display names").
- Use formatter objects in Core Foundation and Cocoa to format numbers, currencies, dates, and times based on the current locale.

For details on how to support localized versions of your software, see *Internationalization Programming Topics*. For information on Core Foundation formatters, see *Data Formatting Guide for Core Foundation*.

Software Configuration

Mac OS X programs commonly use property list files (also known as plist files) to store configuration data. A property list is a text or binary file used to manage a dictionary of key-value pairs. Applications use a special type of property list file, called an information property list (`Info.plist`) file, to communicate key attributes of the application to the system, such as the application's name, unique identification string, and version information. Applications also use property list files to store user preferences or other custom configuration data. If your application stores custom configuration data, you should consider using property lists files as well.

The advantage of property list files is that they are easy to edit and modify from outside the runtime environment of your application. Mac OS X provides several tools for creating and modifying property list files. The Property List Editor application that comes with Xcode is the main application for editing the contents of property lists. Xcode also provides a custom interface for editing your application's `Info.plist` file. (For information about information property lists files and the keys you put in them, see *Runtime Configuration Guidelines*.)

Inside your program, you can read and write property list files programmatically using facilities found in both Core Foundation and Cocoa. For more information on creating and using property lists programmatically, see *Property List Programming Guide* or *Property List Programming Topics for Core Foundation*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Fast User Switching

Introduced in Mac OS X version 10.3, fast user switching lets multiple users share physical access to a single computer without logging out. Only one user at a time can access the computer using the keyboard, mouse, and display; however, one user's session can continue to run while another user accesses the computer. The users can then trade access to the computer and toggle sessions back and forth without disturbing each other's work.

When fast user switching is enabled, an application must be careful not to do anything that might affect another version of that application running in a different user's session. In particular, your application should avoid using or creating any shared resources unless those resources are associated with a particular user session. As you design your application, make sure that any shared resources you use are protected appropriately. For more information on how to do this, see *Multiple User Environments*.

Spaces

Introduced in Mac OS X version 10.5, Spaces lets the user organize windows into groups and switch back and forth between groups to avoid cluttering up the desktop. Most application windows appear in only one space at a time, but there may be times when you need to share a window among multiple spaces. For example, if your application has a set of shared floating palettes, you might need those palettes to show up in every space containing your application's document windows.

Cocoa provides support for sharing windows across spaces through the use of collection behavior attributes on the window. For information about setting these attributes, see *NSWindow Class Reference*.

Accessibility

Millions of people have some type of disability or special need. Federal regulations in the United States stipulate that computers used in government or educational settings must provide reasonable access for people with disabilities. Mac OS X includes built-in functionality to accommodate users with special needs. It also provides software developers with the functions they need to support accessibility in their own applications.

Applications that use Cocoa or modern Carbon interfaces receive significant support for accessibility automatically. For example, applications get the following support for free:

- Zoom features let users increase the size of onscreen elements.
- Sticky keys let users press keys sequentially instead of simultaneously for keyboard shortcuts.
- Mouse keys let users control the mouse with the numeric keypad.
- Full keyboard access mode lets users complete any action using the keyboard instead of the mouse.
- Speech recognition lets users speak commands rather than type them.
- Text-to-speech reads text to users with visual disabilities.
- VoiceOver provides spoken user interface features to assist visually impaired users.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

If your application is designed to work with assistive devices (such as screen readers), you may need to provide additional support. Both Cocoa and Carbon integrate support for accessibility protocols in their frameworks; however, there may still be times when you need to provide additional descriptions or want to change descriptions associated with your windows and controls. In those situations, you can use the appropriate accessibility interfaces to change the settings.

For more information about accessibility, see *Accessibility Overview*.

AppleScript

Mac OS X employs AppleScript as the primary language for making applications scriptable. AppleScript is supported in all application environments as well as in the Classic compatibility environment. Thus, users can write scripts that link together the services of multiple scriptable applications across different environments.

When designing new applications, you should consider AppleScript support early in the process. The key to good AppleScript design is choosing an appropriate data model for your application. The design must not only serve the purposes of your application but should also make it easy for AppleScript implementers to manipulate your content. Once you settle on a model, you can implement the Apple event code needed to support scripting.

For information about AppleScript in Mac OS X, go to http://www.apple.com/applescript. For developer documentation explaining how to support AppleScript in your programs, see Reference Library > Scripting & Automation.

System Applications

Mac OS X provides many applications to help both developers and users implement their projects. A default Mac OS X installation includes an `Applications` directory containing many user and administrative tools that you can use in your development. In addition, there are two special applications that are relevant to running programs: the Finder and the Dock. Understanding the purpose of these applications can help when it comes to designing your own applications.

The Finder

The Finder has many functions in the operating system:

- It is the primary file browser. As such, it is the first tool users see, and one they use frequently to find applications and other files.
- It provides an interface for Spotlight—a powerful search tool for finding files not easily found by browsing.
- It provides a way to access servers and other remote volumes, including a user's iDisk.
- It determines the application in which to open a document when a user double-clicks a document icon.
- It allows users to create file archives.
- It provides previews of images, movies, and sounds in its preview pane.
- It lets users burn content onto CDs and DVDs.
- It provides an AppleScript interface for manipulating files and the Finder user interface.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Keep the Finder in mind as you design your application's interface. Understand that any new behaviors you introduce should follow patterns users have grown accustomed to in their use of the Finder. Although some of the functionality of the Finder, like file browsing, is replicated through the Carbon and Cocoa frameworks, the Finder may be where users feel most comfortable performing certain functions. Your application should interact with the Finder gracefully and should communicate changes to the Finder where appropriate. For example, you might want to embed content by allowing users to drag files from the Finder into a document window of your application.

Another way your application interacts with the Finder is through configuration data. The information property list of your bundled application communicates significant information about the application to the Finder. Information about your application's identity and the types of documents it supports are all part of the information property list file.

For information about the Finder and its relationship to the file system, see *File System Overview*.

The Dock

Designed to help prevent onscreen clutter and aid in organizing work, the always available Dock displays an icon for each open application and minimized document. It also contains icons for commonly used applications and for the Trash. Applications can use the Dock to convey information about the application and its current state.

For guidelines on how to work with the Dock within your program, see *Apple Human Interface Guidelines*. For information on how to manipulate Dock tiles in a Carbon application, see *Dock Tile Programming Guide* and *Application Manager Reference*. To manipulate Dock tiles from a Cocoa application, use the methods of the `NSApplication` and `NSWindow` classes.

Dashboard

Introduced in Mac OS X v10.4, Dashboard provides a lightweight desktop environment for running widgets. Widgets are lightweight web applications that display information a user might use occasionally. You can write widgets to track stock quotes, view the current time, or access key features of a frequently used application. Widgets reside in the Dashboard layer, which is activated by the user and comes into the foreground in a manner similar to Exposé. Mac OS X comes with several standard widgets, including a calculator, clock, and iTunes controller.

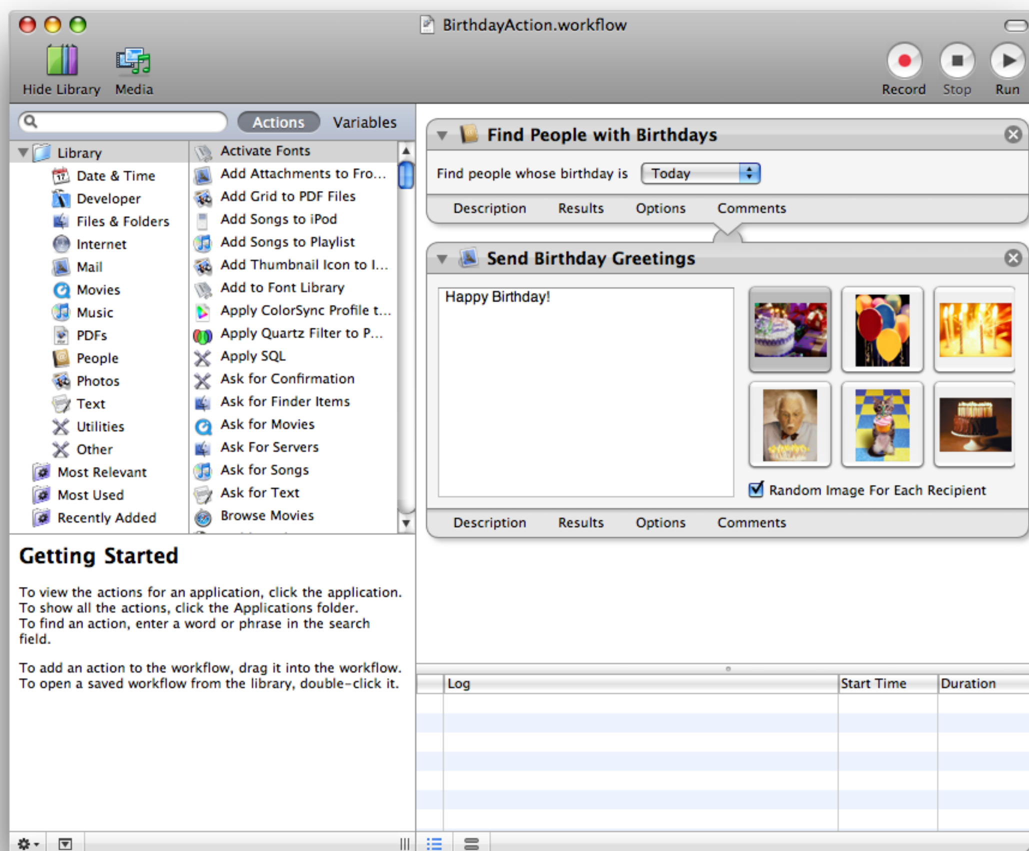
For information about developing Dashboard widgets, see "Dashboard Widgets" (page 85).

Automator

Introduced in Mac OS X version 10.4, Automator lets you automate common workflows on your computer without writing any code. The workflows you create can take advantage of many features of Mac OS X and any standard applications for which predefined **actions** are available. Actions are building blocks that represent tangible tasks, such as opening a file, saving a file, applying a filter, and so on. The output from one action becomes the input to another and you assemble the actions graphically with the Automator application. Figure 5-1 shows the Automator main window and a workflow containing some actions.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Figure 5-1 Automator main window



In cases where actions are not available for the tasks you want, you can often create them yourself. Automator supports the creation of actions using Objective-C code or AppleScript. You can also create actions that are based on shell scripts, Perl, and Python.

In Mac OS X v10.5 and later, Automator supports the “Watch Me Do” feature, which lets you build an action by recording your interactions with Mac OS X and any open applications. You can use workflow variables as placeholders for dynamically changing values or pieces of text in your script. You can also integrate workflows into your applications using the classes of the Automator framework.

For more information about using Automator, see the Automator Help. For information on how to create new Automator actions, see *Automator Programming Guide*. For information about how to integrate workflows into your applications, see the classes in *Automator Framework Reference*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Time Machine

Introduced in Mac OS X v10.5, Time Machine is an application that automatically and transparently backs up the user's files to a designated storage system. Time Machine integrates with the Finder to provide an intuitive interface for locating lost or old versions of files quickly and easily. Time Machine also provides an interface that applications can use to exclude files that should not be backed up. For more information on using this interface, see "Time Machine Support" (page 68).

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Software Development Overview

There are many ways to create an application in Mac OS X. There are also many types of software that you can create besides applications. The following sections introduce the types of software you can create in Mac OS X and when you might consider doing so.

Applications

Applications are by far the predominant type of software created for Mac OS X, or for any platform. Mac OS X provides numerous environments for developing applications, each of which is suited for specific types of development. For information about these environments and the technologies you can use to build your applications, see "Application Technologies" (page 55).

Important: You should always create universal binaries for Carbon, Cocoa, and BSD applications. Java and WebObjects may also need to create universal binaries for bridged code. For information on how to create universal binaries, see *Universal Binary Programming Guidelines, Second Edition*.

Frameworks

A framework is a special type of bundle used to distribute shared resources, including library code, resource files, header files, and reference documentation. Frameworks offer a more flexible way to distribute shared code that you might otherwise put into a dynamic shared library. Whereas image files and localized strings for a dynamic shared library would normally be installed in a separate location from the library itself, in a framework they are integral to the framework bundle. Frameworks also have a version control mechanism that makes it possible to distribute multiple versions of a framework in the same framework bundle.

Apple uses frameworks to distribute the public interfaces of Mac OS X. You can use frameworks to distribute public code and interfaces created by your company. You can also use frameworks to develop private shared libraries that you can then embed in your applications.

Note: Mac OS X also supports the concept of "umbrella" frameworks, which encapsulate multiple subframeworks in a single package. However, this mechanism is used primarily for the distribution of Apple software. The creation of umbrella frameworks by third-party developers is not recommended.

You can develop frameworks using any programming language you choose; however, it is best to choose a language that makes it easy to update the framework later. Apple frameworks generally export programmatic interfaces in either ANSI C or Objective-C. Both of these languages have a well-defined export structure that makes it easy to maintain compatibility between different revisions of the framework. Although it is possible to use other languages when creating frameworks, you may run into binary compatibility problems later when you update your framework code.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

For information on the structure and composition of frameworks, see *Framework Programming Guide*. That document also contains details on how to create public and private frameworks with Xcode.

Important: You should always create universal binaries for frameworks written with Carbon, Cocoa, or BSD APIs. For information on how to create universal binaries, see *Universal Binary Programming Guidelines, Second Edition*.

Plug-ins

Plug-ins are the standard way to extend many applications and system behaviors. Plug-ins are bundles whose code is loaded dynamically into the runtime of an application. Because they are loaded dynamically, they can be added and removed by the user.

There are many opportunities for developing plug-ins for Mac OS X. Developers can create plug-ins for third-party applications or for Mac OS X itself. Some parts of Mac OS X define plug-in interfaces for extending the basic system behavior. The following sections list many of these opportunities for developers, although other software types may also use the plug-in model.

Important: With the transition to Intel-based processors, developers should always create universal binaries for plug-ins written with Carbon, Cocoa, or BSD APIs. For information on how to create universal binaries, see *Universal Binary Programming Guidelines, Second Edition*.

Address Book Action Plug-Ins

An Address Book action plug-in lets you populate the pop-up menus of the Address Book application with custom menu items that use Address Book data to trigger a specific event. For example, you could add an action to a phone number field to trigger the dialing of the number using a Bluetooth-enabled phone.

Address Book action plug-ins are best suited for developers who want to extend the behavior of the Address Book application to support third-party hardware or software. For more information on creating an Address Book action plug-in, see the documentation for the `ABActionDelegate` class.

Application Plug-Ins

Several applications, including iTunes, Final Cut Pro, and Final Cut Express, use plug-ins to extend the features available from the application. You can create plug-ins to implement new effects for these applications or for other applications that support a plug-in model. For information about developing plug-ins for Apple applications, visit the ADC website at http://developer.apple.com/.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Automator Plug-Ins

Introduced in Mac OS X version 10.4, Automator is a workflow-based application that lets users assemble complex scripts graphically using a palette of available actions. You can extend the default set of actions by creating Automator plug-ins to support new actions. Because they can be written in AppleScript or Objective-C, you can write plug-ins for your own application's features or for the features of other scriptable applications.

If you are developing an application, you should think about providing Automator plug-ins for your application's most common tasks. AppleScript is one of the easiest ways for you to create Automator plug-ins because it can take advantage of existing code in your application. If you are an Objective-C developer, you can also use that language to create plug-ins.

For information on how to write an Automator plug-in, see *Automator Programming Guide*.

Contextual Menu Plug-Ins

The Finder associates contextual menus with file-system items to give users a way to access frequently used commands quickly. Third-party developers can extend the list of commands found on these menus by defining their own contextual menu plug-ins. You might use this technique to make frequently used features available to users without requiring them to launch your application. For example, an archiving program might provide commands to compress a file or directory.

The process for creating a contextual menu plug-in is similar to that for creating a regular plug-in. You start by defining the code for registering and loading your plug-in, which might involve creating a factory object or explicitly specifying entry points. To implement the contextual menu behavior, you must then implement several callback functions defined by the Carbon Menu Manager for that purpose. Once complete, you install your plug-in in the `Library/Contextual Menu Items` directory at the appropriate level of the system, usually the local or user level.

For information on how to create a plug-in, see *Plug-ins*. For information on the Carbon Menu Manager functions you need to implement, see *Menu Manager Reference*.

Core Audio Plug-Ins

The Core Audio system supports plug-ins for manipulating audio streams during most processing stages. You can use plug-ins to generate, process, receive, or otherwise manipulate an audio stream. You can also create plug-ins to interact with new types of audio-related hardware devices.

For an introduction to the Core Audio environment, download the Core Audio SDK from http://developer.apple.com/sdk/ and read the documentation that comes with it. Information is also available in Reference Library > Audio > Core Audio.

Image Units

In Mac OS X version 10.4 and later, you can create **image units** for the Core Image and Core Video technologies. An image unit is a collection of filters packaged together in a single bundle. Each filter implements a specific manipulation for image data. For example, you could write a set of filters that perform different kinds of edge detection and package them as one image unit.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

For more information about Core Image, see *Core Image Programming Guide*.

Input Method Components

An input method component is a code module that processes incoming data and returns an adjusted version of that data. A common example of an input method is an interface for typing Japanese or Chinese characters using multiple keystrokes. The input method processes the user keystrokes and returns the complex character that was intended. Other examples of input methods include spelling checkers and pen-based gesture recognition systems.

Input method components are implemented using the Carbon Component Manager. An input method component provides the connection between Mac OS X and any other programs your input method uses to process the input data. For example, you might use a background application to record the input keystrokes and compute the list of potential complex characters that those keystrokes can create.

In Mac OS X v10.5 and later, you can create input methods using the Input Method Kit (`InputMethodKit.framework`). For information on how to use this framework, see *Input Method Kit Framework Reference*. For information on how to create an input method in earlier versions of Mac OS X, see the *BasicInputMethod* sample code and the *Component Manager Reference*.

Interface Builder Plug-Ins

If you create any custom controls for your application, you can create an Interface Builder plug-in to make those controls available in the Interface Builder design environment. Creating plug-ins for your controls lets you go back and redesign your application's user interface using your actual controls, as opposed to generic custom views. For controls that are used frequently in your application, being able to see and manipulate your controls directly can eliminate the need to build your application to see how your design looks.

In Mac OS X v10.5, you should include plug-ins for any of your custom controls inside the framework bundle that implements those controls. Bundling your plug-in with your framework is not required but does make it easier for users. When the user adds your framework to their Xcode project, Interface Builder automatically scans the framework and loads the corresponding plug-in if it is present. If you did not use a framework for the implementation of your controls, you must distribute the plug-in yourself and instruct users to load it using the Interface Builder preferences window.

For information on how to create plug-ins that support Interface Builder 3.0 and later, see *Interface Builder Plug-In Programming Guide* and *Interface Builder Kit Framework Reference*. For information on how to create plug-ins for earlier versions of Interface Builder, see the header files for Interface Builder framework (`InterfaceBuilder.framework`) or the examples in <*Xcode*>`/Examples/Interface Builder`.

Metadata Importers

In Mac OS X version 10.4 and later, you can create a metadata importer for your application's file formats. Metadata importers are used by Spotlight to gather information about the user's files and build a systemwide index. This index is then used for advanced searching based on more user-friendly information.

If your application defines a custom file format, you should always provide a metadata importer for that file format. If your application relies on commonly used file formats, such as JPEG, RTF, or PDF, the system provides a metadata importer for you.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

For information on creating metadata importers, see *Spotlight Importer Programming Guide*.

QuickTime Components

A QuickTime component is a plug-in that provides services to QuickTime-savvy applications. The component architecture of QuickTime makes it possible to extend the support for new media formats, codecs, and hardware. Using this architecture, you can implement components for the following types of operations:

- Compressing/decompressing media data
- Importing/exporting media data
- Capturing media data
- Generating timing signals
- Controlling movie playback
- Implementing custom video effects, filters, and transitions
- Streaming custom media formats

For an overview of QuickTime components, see *QuickTime Overview*. For information on creating specific component types, see the subcategories in Reference Library > QuickTime.

Safari Plug-ins

Beginning with Mac OS X version 10.4, Safari supports a new plug-in model for tying in additional types of content to the web browser. This new model is based on an Objective-C interface and offers significant flexibility to plug-in developers. In particular, the new model lets plug-in developers take advantage of the Tiger API for modifying DOM objects in an HTML page. It also offers hooks so that JavaScript code can interact with the plug-in at runtime.

Safari plug-in support is implemented through the new WebPlugIn object and related objects defined in Web Kit. For information about how to use these objects, see *WebKit Plug-In Programming Topics* and *WebKit Objective-C Framework Reference*.

Dashboard Widgets

Introduced in Mac OS X version 10.4 and later, Dashboard provides a lightweight desktop layer for running **widgets**. Widgets are lightweight web applications that display information a user might use occasionally. You could write widgets to track stock quotes, view the current time, or access key features of a frequently used application. Widgets reside in the Dashboard layer, which is activated by the user and comes into the foreground in a manner similar to Exposé. Mac OS X comes with several standard widgets, including a calculator, clock, and iTunes controller.

Creating widgets is simpler than creating most applications because widgets are effectively HTML-based applications with optional JavaScript code to provide dynamic behavior. Dashboard uses the Web Kit to provide the environment for displaying the HTML and running the JavaScript code. Your widgets can take

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

advantage of several extensions provided by that environment, including a way to render content using Quartz-like JavaScript functions. In Mac OS X v10.5 and later, you can create widgets using the Dashcode application, which is described in "Dashcode" (page 134).

For information on how to create widgets, see *Dashboard Programming Topics*.

Agent Applications

An agent is a special type of application designed to help the user in an unobtrusive manner. Agents typically run in the background, providing information as needed to the user or to another application. Agents can display panels occasionally or come to the foreground to interact with the user if necessary. User interactions should always be brief and have a specific goal, such as setting preferences or requesting a piece of needed information.

An agent may be launched by the user but is more likely to be launched by the system or another application. As a result, agents do not show up in the Dock or the Force Quit window. Agents also do not have a menu bar for choosing commands. User manipulation of an agent typically occurs through dialogs or contextual menus in the agent user interface. For example, the iChat application uses an agent to communicate with the chat server and notify the user of incoming chat requests. The Dock is another agent program that is launched by the system for the benefit of the user.

The way to create an agent application is to create a bundled application and include the `LSUIElement` key in its `Info.plist` file. The `LSUIElement` key notifies the Dock that it should treat the application as an agent when double-clicked by the user. For more information on using this key, see *Runtime Configuration Guidelines*.

Screen Savers

Screen savers are small programs that take over the screen after a certain period of idle activity. Screen savers provide entertainment and also prevent the screen image from being burned into the surface of a screen permanently. Mac OS X supports both slideshows and programmatically generated screen-saver content.

Slideshows

A slideshow is a simple type of screen saver that does not require any code to implement. To create a slideshow, you create a bundle with an extension of `.slideSaver`. Inside this bundle, you place a Resources directory containing the images you want to display in your slideshow. Your bundle should also include an information property list that specifies basic information about the bundle, such as its name, identifier string, and version.

Mac OS X includes several slideshow screen savers you can use as templates for creating your own. These screen savers are located in `/System/Library/Screen Savers`. You should put your own slideshows in either `/Library/Screen Savers` or in the `~/Library/Screen Savers` directory of a user.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Programmatic Screen Savers

A programmatic screen saver is one that continuously generates content to appear on the screen. You can use this type of screen saver to create animations or to create a screen saver with user-configurable options. The bundle for a programmatic screen saver ends with the `.saver` extension.

You create programmatic screen savers using Cocoa and the Objective-C language. Specifically, you create a custom subclass of ScreenSaverView that provides the interface for displaying the screen saver content and options. The information property list of your bundle provides the system with the name of your custom subclass.

For information on creating programmatic screen savers, see *Screen Saver Framework Reference*.

Important: You should always create universal binaries for program-based screensavers written with Carbon, Cocoa, or BSD APIs. For information on how to create universal binaries, see *Universal Binary Programming Guidelines, Second Edition*.

Services

Services are not separate programs that you write; instead, they are features exported by your application for the benefit of other applications. Services let you share the resources and capabilities of your application with other applications in the system.

Services typically act on the currently selected data. Upon initiation of a service, the application that holds the selected data places it on the pasteboard. The application whose service was selected then takes the data, processes it, and puts the results (if any) back on the pasteboard for the original application to retrieve. For example, a user might select a folder in the Finder and choose a service that compresses the folder contents and replaces them with the compressed version. Services can represent one-way actions as well. For example, a service could take the currently selected text in a window and use it to create the content of a new email message.

For information on how to implement services in your Cocoa application, see *System Services*. For information on how to implement services in a Carbon application, see *Setting Up Your Carbon Application to Use the Services Menu*.

Preference Panes

Preference panes are used primarily to modify system preferences for the current user. Preference panes are implemented as plug-ins and installed in `/Library/PreferencePanes` on the user's system. Application developers can also take advantage of these plug-ins to manage per-user application preferences; however, most applications manage preferences using the code provided by the application environment.

You might need to create preference panes if you create:

- Hardware devices that are user-configurable
- Systemwide utilities, such as virus protection programs, that require user configuration

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

If you are an application developer, you might want to reuse preference panes intended for the System Preferences application or use the same model to implement your application preferences.

Because the interfaces are based on Objective-C, you write preference panes primarily using Cocoa. For more information, see *Preference Panes*.

Important: With the transition to Intel-based processors, developers should always create universal binaries for preference panes. For information on how to create universal binaries, see *Universal Binary Programming Guidelines, Second Edition*.

Web Content

Mac OS X supports a variety of techniques and technologies for creating web content. Dynamic websites and web services offer web developers a way to deliver their content quickly and easily.

In addition to "WebObjects" (page 58) and "Dashboard Widgets" (page 85), the following sections list ways to deliver web content in Mac OS X. For more information about developing web content, see *Getting Started with Internet and Web*.

Dynamic Websites

Mac OS X provides support for creating and testing dynamic content in web pages. If you are developing CGI-based web applications, you can create websites using a variety of scripting technologies, including Perl and PHP. A complete list of scripting technologies is provided in "Scripts" (page 90). You can also create and deploy more complex web applications using JBoss, Tomcat, and WebObjects. To deploy your webpages, use the built-in Apache web server.

Safari, Apple's web browser, provides standards-compliant support for viewing pages that incorporate numerous technologies, including HTML, XML, XHTML, DOM, CSS, Java, and JavaScript. You can also use Safari to test pages that contain multimedia content created for QuickTime, Flash, and Shockwave.

SOAP and XML-RPC

The Simple Object Access Protocol (SOAP) is an object-oriented protocol that defines a way for programs to communicate over a network. XML-RPC is a protocol for performing remote procedure calls between programs. In Mac OS X, you can create clients that use these protocols to gather information from web services across the Internet. To create these clients, you use technologies such as PHP, JavaScript, AppleScript, and Cocoa.

If you want to provide your own web services in Mac OS X, you can use WebObjects or implement the service using the scripting language of your choice. You then post your script code to a web server, give clients a URL, and publish the message format your script supports.

For information on how to create client programs using AppleScript, see *XML-RPC and SOAP Programming Guide*. For information on how to create web services, see *WebObjects Web Services Programming Guide*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Sherlock Channels

In Mac OS X v10.4 and earlier, the Sherlock application was a host for Sherlock channels. A Sherlock channel is a developer-created module that combines web services with an Aqua interface to provide a unique way for users to find information. Sherlock channels combined related, but different, types of information in one window.

Sherlock channels are not supported in Mac OS X v10.5 and later.

Mail Stationery

The Mail application in Mac OS X v10.5 and later supports the creation of email messages using templates. Templates provide the user with prebuilt email messages that can be customized quickly before being sent. Because templates are HTML-based, they can incorporate images and advanced formatting to give the user's email a much more stylish and sophisticated appearance.

Developers and web designers can create custom template packages for external or internal users. Each template consists of an HTML page, property list file, and images packaged together in a bundle, which is then stored in the Mail application's stationery directory. The HTML page and images define the content of the email message and can include drop zones for custom user content. The property list file provides Mail with information about the template, such as its name, ID, and the name of its thumbnail image.

For information about how to create new stationery templates, see *Mail Programming Topics*.

Command-Line Tools

Command-line tools are simple programs that manipulate data using a text-based interface. These tools do not use windows, menus, or other user interface elements traditionally associated with applications. Instead, they run from the command-line environment of the Terminal application. Command-line tools require less explicit knowledge of the system to develop and because of that are often simpler to write than many other types of applications. However, command-line tools usually serve a more technically savvy crowd who are familiar with the conventions and syntax of the command-line interface.

Xcode supports the creation of command-line tools from several initial code bases. For example, you can create a simple and portable tool using standard C or C++ library calls, or a more Mac OS X–specific tool using frameworks such as Core Foundation, Core Services, or Cocoa Foundation.

> **Important:** With the transition to Intel-based processors, developers should always create universal binaries for command-line tools written with Carbon, Cocoa, or BSD APIs. For information on how to create universal binaries, see *Universal Binary Programming Guidelines, Second Edition*.

Command-line tools are ideal for implementing simple programs quickly. You can use them to implement low-level system or administrative tools that do not need (or cannot have) a graphical user interface. For example, a system administrator might use command-line tools to gather status information from an Xserve system. You might also use them to test your program's underlying code modules in a controlled environment.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Note: Daemons are a special type of command-line program that run in the background and provide services to system and user-level programs. Developing daemons is not recommended, or necessary, for most developers.

Launch Items, Startup Items, and Daemons

Launch items and startup items are special programs that launch other programs or perform one-time operations during startup and login periods. Daemons are programs that run continuously and act as servers for processing client requests. You typically use launch items and startup items to launch daemons or perform periodic maintenance tasks, such as checking the hard drive for corrupted information. Launch items run under the `launchd` system process and are supported only in Mac OS X v10.4 and later. Startup items are also used to launch system and user-level processes but are deprecated in current versions of Mac OS X. They may be used to launch daemons and run scripts in Mac OS X v10.3.9 and earlier.

Launch items and startup items should not be confused with the login items found in the Accounts system preferences. Login items are typically agent applications that run within a given user's session and can be configured by that user. Launch items and startup items are not user-configurable.

Few developers should ever need to create launch items or daemons. They are reserved for the special case where you need to guarantee the availability of a particular service. For example, Mac OS X provides a launch item to run the DNS daemon. Similarly, a virus-detection program might install a launch item to launch a daemon that monitors the system for virus-like activity. In both cases, the launch item would run its daemon in the root session, which provides services to all users of the system.

For more information about launch items, startup items, and daemons, see *System Startup Programming Topics*.

Important: With the transition to Intel-based processors, developers should always create universal binaries for launch items written with Carbon, Cocoa, or BSD APIs. For information on how to create universal binaries, see *Universal Binary Programming Guidelines, Second Edition*.

Scripts

A script is a set of text commands that are interpreted at runtime and turned into a sequence of actions. Most scripting languages provide high-level features that make it easy to implement complex workflows very quickly. Scripting languages are often very flexible, letting you call other programs and manipulate the data they return. Some scripting languages are also portable across platforms, so that you can use your scripts anywhere.

Table 6-1 lists many of the scripting languages supported by Mac OS X along with a description of the strengths of each language.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Table 6-1 Scripting language summary

Script language	Description
AppleScript	An English-based language for controlling scriptable applications in Mac OS X. Use it to tie together applications involved in a custom workflow or repetitive job. You can also use AppleScript Studio to create standalone applications whose code consists primarily of scripts. See Getting Started With AppleScript for more information.
`bash`	A Bourne-compatible shell script language used to build programs on UNIX-based systems.
`csh`	The C shell script language used to build programs on UNIX-based systems.
Perl	A general-purpose scripting language supported on many platforms. It comes with an extensive set of features suited for text parsing and pattern matching and also has some object-oriented features. See http://www.perl.org/ for more information.
PHP	A cross-platform, general-purpose scripting language that is especially suited for web development. See http://www.php.net/ for more information.
Python	A general-purpose, object-oriented scripting language implemented for many platforms. See http://www.python.org/ for more information. In Mac OS X v10.4 and later, you can also use Python with the Cocoa scripting bridge; see *Ruby and Python Programming Topics for Mac OS X*.
Ruby	A general-purpose, object-oriented scripting language implemented for many platforms. See http://www.ruby-lang.org/ for more information. In Mac OS X v10.5 and later, you can also use Ruby with the Cocoa scripting bridge; see *Ruby and Python Programming Topics for Mac OS X*.
`sh`	The Bourne shell script language used to build programs on UNIX-based systems.
Tcl	Tool Command Language. A general-purpose language implemented for many platforms. It is often used to create graphical interfaces for scripts. See http://www.tcl.tk/ for more information.
`tcsh`	A variant of the C shell script language used to build programs on UNIX-based systems.
`zsh`	The Z shell script language used to build programs on UNIX-based systems.

For introductory material on using the command line, see "Command Line Primer" (page 109).

Scripting Additions for AppleScript

A scripting addition is a way to deliver additional functionality for AppleScript scripts. It extends the basic AppleScript command set by adding systemwide support for new commands or data types. Developers who need features not available in the current command set can use scripting additions to implement those features and make them available to all programs. For example, one of the built-in scripting additions extends the basic file-handling commands to support the reading and writing of file contents from an AppleScript script.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

For information on how to create a scripting addition, see Technical Note TN1164, "Native Scripting Additions."

Important: With the transition to Intel-based processors, developers should always create universal binaries for scripting additions written with Carbon, Cocoa, or BSD APIs. For information on how to create universal binaries, see *Universal Binary Programming Guidelines, Second Edition*.

Kernel Extensions

Most developers have little need to create kernel extensions. Kernel extensions are code modules that load directly into the kernel process space and therefore bypass the protections offered by the Mac OS X core environment. The situations in which you might need a kernel extension are the following:

- Your code needs to handle a primary hardware interrupt.
- The client of your code is inside the kernel.
- A large number of applications require a resource your code provides. For example, you might implement a file-system stack using a kernel extension.
- Your code has special requirements or needs to access kernel interfaces that are not available in the user space.

Kernel extensions are typically used to implement new network stacks or file systems. You would not use kernel extensions to communicate with external devices such as digital cameras or printers. (For information on communicating with external devices, see "Device Drivers" (page 92).)

Note: Beginning with Mac OS X version 10.4, the design of the kernel data structures is changing to a more opaque access model. This change makes it possible for kernel developers to write nonfragile kernel extensions—that is, kernel extensions that do not break when the kernel data structures change. Developers are highly encouraged to use the new API for accessing kernel data structures.

For information about writing kernel extensions, see *Kernel Programming Guide*.

Important: With the transition to Intel-based processors, developers should always create universal binaries for kernel extensions. For information on how to create universal binaries, see *Universal Binary Programming Guidelines, Second Edition*.

Device Drivers

Device drivers are a special type of kernel extension that enable Mac OS X to communicate with all manner of hardware devices, including mice, keyboards, and FireWire drives. Device drivers communicate hardware status to the system and facilitate the transfer of device-specific data to and from the hardware. Mac OS X provides default drivers for many types of devices, but these may not meet the needs of all developers.

Although developers of mice and keyboards may be able to use the standard drivers, many other developers require custom drivers. Developers of hardware such as scanners, printers, AGP cards, and PCI cards typically have to create custom drivers for their devices. These devices require more sophisticated data handling than

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

is usually needed for mice and keyboards. Hardware developers also tend to differentiate their hardware by adding custom features and behavior, which makes it difficult for Apple to provide generic drivers to handle all devices.

Apple provides code you can use as the basis for your custom drivers. The I/O Kit provides an object-oriented framework for developing device drivers using C++. For information on developing device drivers, see *I/O Kit Fundamentals*.

Important: With the transition to Intel-based processors, developers should always create universal binaries for device drivers. For information on how to create universal binaries, see *Universal Binary Programming Guidelines, Second Edition*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Choosing Technologies to Match Your Design Goals

Mac OS X has many layers of technology. Before choosing a specific technology to implement a solution, think about the intended role for that technology. Is that technology appropriate for your needs? Is there a better technology available? In some cases, Mac OS X offers several technologies that implement the same behavior but with varying levels of complexity and flexibility. Understanding your operational needs can help you make appropriate choices during design.

As you consider the design of your software, think about your overall goals. The following sections list some of the high-level goals you should strive for in your Mac OS X software. Along with each goal are a list of some technologies that can help you achieve that goal. These lists are not exhaustive but provide you with ideas you might not have considered otherwise. For specific design tips related to these goals, see *Apple Human Interface Guidelines*.

High Performance

Performance is the perceived measure of how fast or efficient your software is, and it is critical to the success of all software. If your software seems slow, users may be less inclined to buy it. Even software that uses the most optimal algorithms may seem slow if it spends more time processing data than responding to the user.

Developers who have experience programming on other platforms (including Mac OS 9) should take the time to learn about the factors that influence performance on Mac OS X. Understanding these factors can help you make better choices in your design and implementation. For information about performance factors and links to performance-related documentation, see *Performance Overview*.

Table 7-1 lists several Mac OS X technologies that you can use to improve the performance of your software.

Table 7-1 Technologies for improving performance

Technology	Description
Universal Binaries	Because Macintosh computers may contain PowerPC or Intel processors, it is important to create universal binaries to ensure that your code always runs natively on the user's computer. Native code runs much faster than emulated code and the Mach-O file format can easily accommodate multiple copies of your executable code in a single package. For information on how to create universal binaries, see *Universal Binary Programming Guidelines, Second Edition*.
64-bit	Although not appropriate in all cases, providing a 64-bit version of your application can improve performance, especially on Intel-based Macintosh computers. The 64-bit capable Intel processors typically have more hardware registers available for performing calculations and passing function parameters. More registers often leads to better performance. As always, test your code in both 32-bit and 64-bit modes to see if providing a 64-bit version is worthwhile. For more information, see *64-Bit Transition Guide*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Technology	Description
`NSOperation` and `NSOperationQueue`	Mac OS X v10.5 includes two new Cocoa classes that simplify the process of supporting multiple threads in your application. The `NSOperation` object acts as a wrapper for encapsulated tasks while the `NSOperationQueue` object manages the execution of those tasks. Operations support dependency and priority ordering and can be customized to configure the threading environment as needed. For more information about these classes, see *Threading Programming Guide*.
Threads	With all new Macintosh computers shipping with multi-core processors, one way to take advantage of the extra computing power of these machines is to use threads to exploit any inherent parallelism in your software. Mac OS X implements user-level threads using the POSIX threading package but also supports several higher-level APIs for managing threads. For information about these APIs and threading support in general, see "Threading Support" (page 28) and *Threading Programming Guide*.
Instruments and Shark	Apple provides a suite of performance tools for measuring many aspects of your software. Instruments and Shark in particular provide new ways of looking at your application while it runs and analyzing its performance. Use these tools to identify hot spots and gather performance metrics that can help identify potential problems. For more information Instruments, see "Instruments" (page 135). For information about Shark and the other performance tools that come with Mac OS X, see "Performance Tools" (page 148).
Accelerate Framework	The Accelerate framework provides an API for performing multiple scalar or floating-point operations in parallel by taking advantage of the underlying processor's vector unit. Because it is tuned for both PowerPC and Intel processor architectures, using the Accelerate framework eliminates the need for you to write custom code for both the AltiVec and SSE vector units. For more information about using this framework, see *Accelerate Release Notes*.
Lower-level APIs	Mac OS X provides many layers of APIs. As you consider the design of your application, examine the available APIs to find the appropriate tradeoff between performance, simplicity, and flexibility that you need. Usually, lower-level system APIs offer the best performance but are more complicated to use. Conversely, higher-level APIs may be simpler to use but be less flexible. Whenever possible, choose the lowest-level API that you feel comfortable using.

Mac OS X supports many modern and legacy APIs. Most of the legacy APIs derive from the assorted managers that were part of the original Macintosh Toolbox and are now a part of Carbon. While many of these APIs still work in Mac OS X, they are not as efficient as APIs created specifically for Mac OS X. In fact, many APIs that provided the best performance in Mac OS 9 now provide the worst performance in Mac OS X because of fundamental differences in the two architectures.

Note: For specific information about legacy Carbon managers and the recommended replacements for them, see "Carbon Considerations" (page 105).

As Mac OS X evolves, the list of APIs and technologies it encompasses may change to meet the needs of developers. As part of this evolution, less efficient interfaces may be deprecated in favor of newer ones. Apple makes these changes only when deemed absolutely necessary and uses the availability macros (defined in `/usr/include/AvailabilityMacros.h`) to identify deprecated interfaces. When you compile your code,

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

deprecated interfaces also trigger the generation of compiler warnings. Use these warnings to find deprecated interfaces, and then check the corresponding reference documentation or header files to see if there are recommended replacements.

Easy to Use

An easy-to-use program offers a compelling, intuitive experience for the user. It offers elegant solutions to complex problems and has a well thought out interface that uses familiar paradigms. It is easy to install and configure because it makes intelligent choices for the user, but it also gives the user the option to override those choices when needed. It presents the user with tools that are relevant in the current context, eliminating or disabling irrelevant tools. It also warns the user against performing dangerous actions and provides ways to undo those actions if taken.

Table 7-2 lists several Mac OS X technologies that you can use to make your software easier to use.

Table 7-2 Technologies for achieving ease of use

Technology	Description
Aqua	If your program has a visual interface, it should adhere to the human interface guidelines for Aqua, which include tips for how to lay out your interface and manage its complexity. For more information, see "Aqua" (page 71).
Quick Look	Introduced in Mac OS X v10.5, Quick Look generates previews of user documents that can be displayed in the Finder and Dock. These previews make it easier for the user to find relevant information quickly without launching any applications. For more information, see "Quick Look" (page 71).
Bonjour	Bonjour simplifies the process of configuring and detecting network services. Your program can vend network services or use Bonjour to be a client of an existing network service. For more information, see "Bonjour" (page 59).
Accessibility technologies	The Accessibility interfaces for Carbon and Cocoa make it easier for people with disabilities to use your software. For more information, see "Accessibility" (page 75).
AppleScript	AppleScript makes it possible for users to automate complex workflows quickly. It also gives users a powerful tool for controlling your application. For more information, see "AppleScript" (page 76).
Internationalization	Mac OS X provides significant infrastructure for internationalizing software bundles. For more information, see "Internationalization and Localization" (page 74).
Keychain Services	Keychains provide users with secure access to passwords, certificates, and other secret information. Adding support for Keychain Services in your program can reduce the number of times you need to prompt the user for passwords and other secure information. For more information, see "Keychain Services" (page 63).

For information on designing an easy-to-use interface, see *Apple Human Interface Guidelines*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Attractive Appearance

One feature that draws users to the Macintosh platform, and to Mac OS X in particular, is the stylish design and attractive appearance of the hardware and software. Although creating attractive hardware and system software is Apple's job, you should take advantage of the strengths of Mac OS X to give your own programs an attractive appearance.

The Finder and other programs that come with Mac OS X use high-resolution, high-quality graphics and icons that include 32-bit color and transparency. You should make sure that your programs also use high-quality graphics both for the sake of appearance and to better convey relevant information to users. For example, the system uses pulsing buttons to identify the most likely choice and transparency effects to add a dimensional quality to windows.

Table 7-3 lists several Mac OS X technologies you can use to ensure that your software has an attractive appearance.

Table 7-3 Technologies for achieving an attractive appearance

Technology	Description
Aqua	Aqua defines the guidelines all developers should follow when crafting their application's user interface. Following these guidelines ensures that your application looks and feels like a Mac OS X application. For more information, see "Aqua" (page 71).
Resolution independence	Screen resolutions continue to increase with most screens now supporting over 100 pixels per inch. In order to prevent content from shrinking too much, Mac OS X will soon apply a scaling factor to drawing operations to keep them at an appropriate size. Your software needs to be ready for this scaling factor by being able to draw more detailed content in the same "logical" drawing area. For more information, see "Resolution-Independent User Interface" (page 72).
Core Animation	In Mac OS X v10.5 and later, you can use Core Animation to add advanced graphics behaviors to your software. Core Animation a lightweight mechanism for performing advanced animations in your Cocoa views. For more information, see "Core Animation" (page 44).
Quartz	Quartz is the native (and preferred) 2D rendering API for Mac OS X. It provides primitives for rendering text, images, and vector shapes and includes integrated color management and transparency support. For more information, see "Quartz" (page 41).
Core Text	In Mac OS X v10.5 and later, Core Text replaces the ATSUI and MLTE technologies as the way to high quality rendering and layout of Unicode text for Carbon and Cocoa applications. The Cocoa text system uses Core Text for its implementation. For more information, see "Core Text" (page 47).
Core Image	In Mac OS X v10.4 and later, Core Image provides advanced image processing effects for your application. Core Image makes it possible to manipulate image data in real time using the available hardware rendering and to perform complex manipulations that make your application look stunning. For more information, see "Core Image" (page 45).

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Technology	Description
OpenGL	OpenGL is the preferred 3D rendering API for Mac OS X. The Mac OS X implementation of OpenGL is hardware accelerated on many systems and has all of the standard OpenGL support for shading and textures. See *OpenGL Programming Guide for Mac OS X* for an overview of OpenGL and guidelines on how to use it. For an example of how to use OpenGL with Cocoa, see the sample code project *Cocoa OpenGL*.

Reliability

A reliable program is one that earns the user's trust. Such a program presents information to the user in an expected and desired way. A reliable program maintains the integrity of the user's data and does everything possible to prevent data loss or corruption. It also has a certain amount of maturity to it and can handle complex situations without crashing.

Reliability is important in all areas of software design, but especially in areas where a program may be running for an extended period of time. For example, scientific programs often perform calculations on large data sets and can take a long time to complete. If such a program were to crash during a long calculation, the scientist could lose days or weeks worth of work.

As you start planning a new project, put some thought into what existing technologies you can leverage from both Mac OS X and the open-source community. For example, if your application displays HTML documents, it doesn't make sense to write your own HTML parsing engine when you can use the Web Kit framework instead.

By using existing technologies, you reduce your development time by reducing the amount of new code you have to write and test. You also improve the reliability of your software by using code that has already been designed and tested to do what you need.

Using existing technologies has other benefits as well. For many technologies, you may also be able to incorporate future updates and bug fixes for free. Apple provides periodic updates for many of its shipping frameworks and libraries, either through software updates or through new versions of Mac OS X. If your application links to those frameworks, it receives the benefit of those updates automatically.

All of the technologies of Mac OS X offer a high degree of reliability. However, Table 7-4 lists some specific technologies that improve reliability by reducing the amount of complex code you have to write from scratch.

Table 7-4 Technologies for achieving reliability

Technology	Description
Code signing	Code signing associates a digital signature with your application and helps the system determine when your application has changed, possibly because of tampering. When changes occur, the system can warn the user and provide an option for disabling the application. For more information, see "Code Signing" (page 73).
Authorization Services	Authorization Services provides a way to ensure that only authorized operations take place. Preventing unauthorized access helps protect your program as well as the rest of the system. See *Authorization Services Programming Guide* for more information.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Technology	Description
Core Foundation	Core Foundation supports basic data types and eliminates the need for you to implement string and collection data types, among others. Both Carbon and Cocoa support the Core Foundation data types, which makes it easier for you to integrate them into your own data structures. See *Getting Started with Core Foundation* for more information.
Web Kit	The Web Kit provides a reliable, standards-based mechanism for rendering HTML content (including JavaScript code) in your application.

Adaptability

An adaptable program is one that adjusts appropriately to its surroundings; that is, it does not stop working when the current conditions change. If a network connection goes down, an adaptable program lets the user continue to work offline. Similarly, if certain resources are locked or become unavailable, an adaptable program finds other ways to meet the user's request.

One of the strengths of Mac OS X is its ability to adapt to configuration changes quickly and easily. For example, if the user changes a computer's network configuration from the system preferences, the changes are automatically picked up by applications such as Safari and Mail, which use CFNetwork to handle network configuration changes automatically.

Table 7-5 lists some Mac OS X technologies that you can use to improve the overall adaptability of your software.

Table 7-5 Technologies for achieving adaptability

Technology	Description
FSEvents API	The FSEvents API lets you detect changes to the file system easily and efficiently. You might use this technology to update your application's internal data structures whenever changes occur to specific directories or directory hierarchies. For more information, see "FSEvents API" (page 34).
Core Foundation	Core Foundation provides services for managing date, time, and number formats based on any locale. See Reference Library > Core Foundation for specific reference documents.
Quartz Services	Quartz Services provides access to screen information and provides notifications when screen information changes. See *Quartz Display Services Reference* for more information.
Bonjour	Bonjour simplifies the process of configuring and detecting network services. Your program can vend network services or use Bonjour to be a client of an existing network service. For more information, see "Bonjour" (page 59).
System Configuration	The System Configuration framework provides information about availability of network entities. See *System Configuration Framework Reference* and *System Configuration Programming Guidelines* for more information.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Interoperability

Interoperability refers to a program's ability to communicate across environments. This communication can occur at either the user or the program level and can involve processes on the current computer or on remote computers. At the program level, an interoperable program supports ways to move data back and forth between itself and other programs. It might therefore support the pasteboard and be able to read file formats from other programs on either the same or a different platform. It also makes sure that the data it creates can be read by other programs on the system.

Users see interoperability in features such as the pasteboard (the Clipboard in the user interface), drag and drop, AppleScript, Bonjour, and services in the Services menu. All of these features provide ways for the user to get data into or out of an application.

Table 7-6 lists some Mac OS X technologies that you can use to improve the interoperability of your software.

Table 7-6 Technologies for achieving interoperability

Technology	Description
AppleScript	AppleScript is a scripting system that gives users direct control over your application as well as parts of Mac OS X. See *AppleScript Overview* for information on supporting AppleScript.
Drag and drop	Although primarily implemented in applications, you can add drag and drop support to any program with a user interface. See *Drag Manager Reference* or *Drag and Drop Programming Topics for Cocoa* for information on how to integrate drag and drop support into your program.
Pasteboard	Both Carbon and Cocoa support cut, copy, and paste operations through the pasteboard. See the `Pasteboard.h` header file in the HIServices framework or *Pasteboard Programming Topics for Cocoa* for information on how to support the pasteboard in your program.
Bonjour	Your program can vend network services or use Bonjour to be a client of an existing network service. For more information, see "Bonjour" (page 59).
Services	Services let the user perform a specific operation in your application using data on the pasteboard. Services use the pasteboard to exchange data but act on that data in a more focused manner than a standard copy-and-paste operation. For example, a service might create a new mail message and paste the data into the message body. See *Setting Up Your Carbon Application to Use the Services Menu* or *System Services* for information on setting up an application to use services.
XML	XML is a structured format that can be used for data interchange. Mac OS X provides extensive support for reading, writing, and parsing XML data. For more information, see "XML Parsing Libraries" (page 68).

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Mobility

Designing for mobility has become increasingly important as laptop usage soars. A program that supports mobility doesn't waste battery power by polling the system or accessing peripherals unnecessarily, nor does it break when the user moves from place to place, changes monitor configurations, puts the computer to sleep, or wakes the computer up.

To support mobility, programs need to be able to adjust to different system configurations, including network configuration changes. Many hardware devices can be plugged in and unplugged while the computer is still running. Mobility-aware programs should respond to these changes gracefully. They should also be sensitive to issues such as power usage. Constantly accessing a hard drive or optical drive can drain the battery of a laptop quickly. Be considerate of mobile users by helping them use their computer longer on a single battery charge.

Table 7-7 lists some Mac OS X technologies that you can use to improve the mobility of your software.

Table 7-7 Technologies for achieving mobility

Technology	Description
Performance	An efficient application uses fewer instructions to compute its data. On portable computers, this improved efficiency translates to power savings and a longer battery life. You should strive to make your applications as efficient as possible using the available system technologies and tools. For more information, see "High Performance" (page 95).
CFNetwork	CFNetwork provides a modern interface for accessing network services and handling changes in the network configuration. See *CFNetwork Programming Guide* for an introduction to the CFNetwork API.
Quartz Services	Quartz Services provides access to screen information and provides notifications when screen information changes. See *Quartz Display Services Reference* for information about the API.
Bonjour	Bonjour lets mobile users find services easily or vend their own services for others to use. For more information, see "Bonjour" (page 59).
System Configuration	The System Configuration framework is the foundation for Apple's mobility architecture. You can use its interfaces to get configuration and status information for network entities. It also sends out notifications when the configuration or status changes. See *System Configuration Programming Guidelines* for more information.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Porting Tips

Although many applications have been created from scratch for Mac OS X, many more have been ported from existing Windows, UNIX, or Mac OS 9 applications. With the introduction of the G5 processor, some application developers are even taking the step of porting their 32-bit applications to the 64-bit memory space offered by the new architecture.

The Reference Library > Porting section of the Apple Developer Connection Reference Library contains documents to help you in your porting efforts. The following sections also provide general design guidelines to consider when porting software to Mac OS X.

64-Bit Considerations

With Macintosh computers using 64-bit PowerPC and Intel processors, developers can begin writing software to take advantage of the 64-bit architecture provided by these chips. For many developers, however, compiling their code into 64-bit programs may not offer any inherent advantages. Unless your program needs more than 4 GB of addressable memory, supporting 64-bit pointers may only reduce the performance of your application.

When you compile a program for a 64-bit architecture, the compiler doubles the size of all pointer variables. This increased pointer size makes it possible to address more than 4 GB of memory, but it also increases the memory footprint of your application. If your application does not take advantage of the expanded memory limits, it may be better left as a 32-bit program.

Regardless of whether your program is currently 32-bit or 64-bit, there are some guidelines you should follow to make your code more interoperable with other programs. Even if you don't plan to implement 64-bit support soon, you may need to communicate with 64-bit applications. Unless you are explicit about the data you exchange, you may run into problems. The following guidelines are good to observe regardless of your 64-bit plans.

- Avoid casting pointers to anything but a pointer. Casting a pointer to a scalar value has different results for 32-bit and 64-bit programs. These differences could be enough to break your code later or cause problems when your program exchanges data with other programs.
- Be careful not to make assumptions about the size of pointers or other scalar data types. If you want to know the size of a type, always use the `sizeof` (or equivalent) operator.
- If you write integer values to a file, make sure your file format specifies the exact size of the value. For example, rather than assume the generic type `int` is 32 bits, use the more explicit types `SInt32` or `int32_t`, which are guaranteed to be the correct size.
- If you exchange integer data with other applications across a network, make sure you specify the exact size of the integer.

There are several documents to help you create 64-bit applications. For general information about making the transition, see *64-Bit Transition Guide*. For Cocoa-specific information, see *64-Bit Transition Guide for Cocoa*. For Carbon-specific information, see *64-Bit Guide for Carbon Developers*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

.

Windows Considerations

If you are a Windows developer porting your application to Mac OS X, be prepared to make some changes to your application as part of your port. Applications in Mac OS X have an appearance and behavior that are different from Windows applications in many respects. Unless you keep these differences in mind during the development cycle, your application may look out of place in Mac OS X.

The following list provides some guidelines related to the more noticeable differences between Mac OS X and Windows applications. This list is not exhaustive but is a good starting point for developers new to Mac OS X. For detailed information on how your application should look and behave in Mac OS X, see *Apple Human Interface Guidelines*. For general porting information, see *Porting to Mac OS X from Windows Win32 API*.

- **Avoid custom controls**. Avoid creating custom controls if Mac OS X already provides equivalent controls for your needs. Custom controls are appropriate only in situations where the control is unique to your needs and not provided by the system. Replacing standard controls can make your interface look out of place and might confuse users.
- **Use a single menu bar**. The Mac OS X menu bar is always at the top of the screen and always contains the commands for the currently active application. You should also pay attention to the layout and placement of menu bar commands, especially commonly used commands such as New, Open, Quit, Copy, Minimize, and Help.
- **Pay attention to keyboard shortcuts**. Mac OS X users are accustomed to specific keyboard shortcuts and use them frequently. Do not simply migrate the shortcuts from your Windows application to your Mac OS X application. Also remember that Mac OS X uses the Command key not the Control key as the main keyboard modifier.
- **Do not use MDI**. The Multiple Document Interface (MDI) convention used in Microsoft Windows directly contradicts Mac OS X design guidelines. Windows in Mac OS X are document-centric and not application-centric. Furthermore, the size of a document window is constrained only by the user's desktop size.
- **Use Aqua**. Aqua gives Mac OS X applications the distinctive appearance and behavior that users expect from the platform. Using nonstandard layouts, conventions, or user interface elements can make your application seem unpolished and unprofessional.
- **Design high-quality icons and images**. Mac OS X icons are often displayed in sizes varying from 16x16 to 512x512 pixels. These icons are usually created professionally, with millions of colors and photo-realistic qualities. Your application icons should be vibrant and inviting and should immediately convey your application's purpose.
- **Design clear and consistent dialogs**. Use the standard Open, Save, printing, Colors, and Font dialogs in your applications. Make sure alert dialogs follow a consistent format, indicating what happened, why it happened, and what to do about it.
- **Consider toolbars carefully**. Having a large number of buttons, especially in an unmovable toolbar, contributes to visual clutter and should be avoided. When designing toolbars, include icons only for menu commands that are not easily discoverable or that may require multiple clicks to be reached.
- **Use an appropriate layout for your windows**. The Windows user interface relies on a left-biased, more crowded layout, whereas Aqua relies on a center-biased, spacious layout. Follow the Aqua guidelines to create an appealing and uncluttered interface that focuses on the task at hand.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

- **Avoid application setup steps**. Whenever possible, Mac OS X applications should be delivered as drag-and-drop packages. If you need to install files in multiple locations, use an installation package to provide a consistent installation experience for the user. If your application requires complex setup procedures in order to run, use a standard Mac OS X assistant. For more information, see "Bundles and Packages" (page 73).
- **Use filename extensions**. Mac OS X fully supports and uses filename extensions. For more information about filename extensions, see *File System Overview*.

Carbon Considerations

If you develop your software using Carbon, there are several things you can do to make your programs work better in Mac OS X. The following sections list migration tips and recommendations for technologies you should be using.

Migrating From Mac OS 9

If you were a Mac OS 9 developer, the Carbon interfaces should seem very familiar. However, improvements in Carbon have rendered many older technologies obsolete. The sections that follow list both the required and the recommended replacement technologies you should use instead.

Required Replacement Technologies

The technologies listed in Table 8-1 cannot be used in Carbon. You must use the technology in the "Now use" column instead.

Table 8-1 Required replacements for Carbon

Instead of	Now use
Any device manager	I/O Kit
Apple Guide	Apple Help
AppleTalk Manager	BSD sockets or CFNetwork
Help Manager	Carbon Help Manager
PPC Toolbox	Apple events
Printing Manager	Core Printing Manager
QuickDraw 3D	OpenGL
QuickDraw GX	Quartz and Apple Type Services for Unicode Imaging (ATSUI)
Standard File Package	Navigation Services
Vertical Retrace Manager	Time Manager

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Recommended Replacement Technologies

The technologies listed in Table 8-2 can still be used in Carbon, but the indicated replacements provide more robust support and are preferred.

Table 8-2 Recommended replacements for Carbon

Instead of	Now use
Display Manager	Quartz Services
Event Manager	Carbon Event Manager
Font Manager	Apple Type Services for Fonts
Internet Config	Launch Services and System Configuration
Open Transport	BSD sockets or CFNetwork
QuickDraw	Quartz 2D
QuickDraw Text	Core Text
Resource Manager	Interface Builder Services
Script Manager	Unicode Utilities
TextEdit	Multilingual Text Engine
URL Access Manager	CFNetwork

Use the Carbon Event Manager

Use of the Carbon Event Manager is strongly recommended for new and existing Carbon applications. The Carbon Event Manager provides a more robust way to handle events than the older Event Manager interfaces. For example, the Carbon Event Manager uses callback routines to notify your application when an event arrives. This mechanism improves performance and offers better mobility support by eliminating the need to poll for events.

For an overview of how to use the Carbon Event Manager, see *Carbon Event Manager Programming Guide*.

Use the HIToolbox

The Human Interface Toolbox is the technology of choice for implementing user interfaces with Carbon. The HIToolbox extends the Macintosh Toolbox and offers an object-oriented approach to organizing the content of your application windows. This new approach to user interface programming is the future direction for Carbon and is where new development and improvements are being made. If you are currently using the Control Manager and Window Manager, you should consider adopting the HIToolbox.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Note: The HIToolbox interfaces are available for creating 32-bit applications only. If you are creating 64-bit applications, you should use Cocoa for your user interface instead.

For an overview of HIView and other HIToolbox objects, see the documents in Reference Library > Carbon > Human Interface Toolbox.

Use Nib Files

Nib files, which you create with Interface Builder, are the best way to design your application interface. The design and layout features of Interface Builder will help you create Aqua-compliant windows and menus. Even if you do not plan to load the nib file itself, you can still use the metrics from this file in your application code.

For information about using Interface Builder, see *Interface Builder User Guide*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Command Line Primer

A command-line interface is a way for you to manipulate your computer in situations where a graphical approach is not available. The Terminal application is the Mac OS X gateway to the BSD command-line interface. Each window in Terminal contains a complete execution context, called a **shell**, that is separate from all other execution contexts. The shell itself is an interactive programming language interpreter, with a specialized syntax for executing commands and writing structured programs, called shell scripts. A shell remains active as long as its Terminal window remains open.

Different shells feature slightly different capabilities and programming syntax. Although you can use any shell of your choice, the examples in this book assume that you are using the standard Mac OS X shell. The standard shell is `bash` if you are running Mac OS X v10.3 or later and `tcsh` if you are running an earlier version of the operating system.

The following sections provide some basic information and tips about using the command-line interface more effectively; they are not intended as an exhaustive reference for using the shell environments.

Basic Shell Concepts

Before you start working in any shell environment, there are some basic features of shell programming that you should understand. Some of these features are specific to Mac OS X, but many are common to all platforms that support shell programming.

Getting Information

At the command-line level, most documentation comes in the form of man pages. These are formatted pages that provide reference information for many shell commands, programs, and high-level concepts. To access one of these pages, you type the `man` command followed by the name of the thing you want to look up. For example, to look up information about the `bash` shell, you would type `man bash`. The man pages are also included in the ADC Reference Library. For more information, see *Mac OS X Man Pages*.

Note: Not all commands and programs have man pages. For a list of available man pages, look in the `/usr/share/man` directory.

Most shells have a command or man page that displays the list of built-in commands. Table A-1 lists the available shells in Mac OS X along with the ways you can access the list of built-in commands for the shell.

Table A-1 Getting a list of built-in commands

Shell	Command
`bash`	`help` or `bash`

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Shell	Command
sh	help or sh
csh	builtins or csh
tcsh	builtins or tcsh
zsh	zshbuiltins

Specifying Files and Directories

Most commands in the shell operate on files and directories, the locations of which are identified by paths. The directory names that comprise a path are separated by forward-slash characters. For example, the path to the Terminal program is /Applications/Utilities/Terminal.app.

Table A-2 lists some of the standard shortcuts used to represent specific directories in the system. Because they are based on context, these shortcuts eliminate the need to type full paths in many situations.

Table A-2 Special path characters and their meaning

Path string	Description
.	A single period represents the current directory. This value is often used as a shortcut to eliminate the need to type in a full path. For example, the string "./Test.c" represents the Test.c file in the current directory.
..	Two periods represents the parent directory of the current directory. This string is used for navigating up one level from the current through the directory hierarchy. For example, the string "../Test" represents a sibling directory (named Test) of the current directory.
~	The tilde character represents the home directory of the currently logged-in user. In Mac OS X, this directory either resides in the local /Users directory or on a network server. For example, to specify the Documents directory of the current user, you would specify ~/Documents.

File and directory names traditionally include only letters, numbers, a period (.), or the underscore character (_). Most other characters, including space characters, should be avoided. Although some Mac OS X file systems permit the use of these other characters, including spaces, you may have to add single or double quotation marks around any pathnames that contain them. For individual characters, you can also "escape" the character, that is, put a backslash character (\) immediately before the character in your string. For example, the path name My Disk would become either "My Disk" or My\ Disk.

Accessing Files on Volumes

On a typical UNIX system, the storage provided by local disk drives is coalesced into a single monolithic file system with a single root directory. This differs from the way the Finder presents local disk drives, which is as one or more volumes, with each volume acting as the root of its own directory hierarchy. To satisfy both worlds, Mac OS X includes a hidden directory Volumes at the root of the local file system. This directory contains all of the volumes attached to the local computer. To access the contents of other local volumes,

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

you should always add the volume path at the beginning of the remaining directory information. For example, to access the `Applications` directory on a volume named `MacOSX`, you would use the path `/Volumes/MacOSX/Applications`

Note: To access files on the boot volume, you are not required to add volume information, since the root directory of the boot volume is `/`. Including the information still works, though, and is consistent with how you access other volumes. You must include the volume path information for all other volumes.

Flow Control

Many programs are capable of receiving text input from the user and printing text out to the console. They do so using the standard pipes (listed in Table A-3), which are created by the shell and passed to the program automatically.

Table A-3 Input and output sources for programs

Pipe	Description
`stdin`	The standard input pipe is the means through which data enters a program. By default, this is data typed in by the user from the command-line interface. You can also redirect the output from files or other commands to `stdin`.
`stdout`	The standard output pipe is where the program output is sent. By default, program output is sent back to the command line. You can also redirect the output from the program to other commands and programs.
`stderr`	The standard error pipe is where error messages are sent. By default, errors are displayed on the command line like standard output.

Redirecting Input and Output

From the command line you may redirect input and output from a program to a file or another program. You use the greater-than (>) character to redirect command output to a file and the less-than (<) character to use a file as input to the program. Redirecting file output lets you capture the results of running the command in the file system and store it for later use. Similarly, providing an input file lets you provide a program with preset input data, instead of requiring the user to type in that data.

In addition to file redirection, you can also redirect the output of one program to the input of another using the vertical bar (|) character. You can combine programs in this manner to implement more sophisticated versions of the same programs. For example, the command `man bash | grep "builtin commands"` redirects the formatted contents of the specified `man` page to the `grep` program, which searches those contents for any lines containing the word "commands". The result is a text listing of only those lines with the specified text, instead of the entire `man` page.

For more information about flow control, see the man page for the shell you are using.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Terminating Programs

To terminate the current running program from the command line, type Control-C. This keyboard shortcut sends an abort signal to the current command. In most cases this causes the command to terminate, although commands may install signal handlers to trap this command and respond differently.

Frequently Used Commands

Shell programming involves a mixture of built-in shell commands and standard programs that run in all shells. While most shells offer the same basic set of commands, there are often variations in the syntax and behavior of those commands. In addition to the shell commands, Mac OS X also provides a set of standard programs that run in all shells.

Table A-4 lists some of the more commonly used commands and programs. Because most of the items in this table are not built-in shell commands, you can use them from any shell. For syntax and usage information for each command, see the corresponding man page. For a more in-depth list of commands and their accompanying documentation, see *Mac OS X Man Pages*.

Table A-4 Frequently used commands and programs

Command	Meaning	Description
`cat`	Catenate	Catenates the specified list of files to `stdout`.
`cd`	Change Directory	A common shell command used to navigate the directory hierarchy.
`cp`	Copy	Copies files and directories (using the -r option) from one location to another.
`date`	Date	Displays the current date and time using the standard format. You can display this information in other formats by invoking the command with specific arguments.
`echo`	Echo to Output	Writes its arguments to `stdout`. This command is most often used in shell scripts to print status information to the user.
`less`	Scroll Through Text	Used to scroll through the contents of a file or the results of another shell command. This command allows forward and backward navigation through the text.
`ls`	List	Displays the contents of the current directory. Specify the `-a` argument to list all directory contents (including hidden files and directories). Use the `-l` argument to display detailed information for each entry.
`mkdir`	Make Directory	Creates a new directory.
`more`	Scroll Through Text	Similar to the `less` command but more restrictive. Allows forward scrolling through the contents of a file or the results of another shell command.
`mv`	Move	Moves files and directories from one place to another. You also use this command to rename files and directories.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Command	Meaning	Description
open	Open an application or file.	You can use this command to launch applications from Terminal and optionally open files in that application.
pwd	Print Working Directory	Displays the full path of the current directory.
rm	Remove	Deletes the specified file or files. You can use pattern matching characters (such as the asterisk) to match more than one file. You can also remove directories with this command, although use of rmdir is preferred.
rmdir	Remove Directory	Deletes a directory. The directory must be empty before you delete it.
Ctrl-C	Abort	Sends an abort signal to the current command. In most cases this causes the command to terminate, although commands may install signal handlers to trap this command and respond differently.

Environment Variables

Some programs require the use of environment variables for their execution. Environment variables are variables inherited by all programs executed in the shell's context. The shell itself uses environment variables to store information such as the name of the current user, the name of the host computer, and the paths to any executable programs. You can also create environment variables and use them to control the behavior of your program without modifying the program itself. For example, you might use an environment variable to tell your program to print debug information to the console.

To set the value of an environment variable, you use the appropriate shell command to associate a variable name with a value. For example, in the bash shell, to set the variable MYFUNCTION to the value MyGetData in the global shell environment you would type the following command in a Terminal window:

```
% export MYFUNCTION=MyGetData
```

When you launch an application from a shell, the application inherits much of its parent shell's environment, including any exported environment variables. This form of inheritance can be a useful way to configure the application dynamically. For example, your application can check for the presence (or value) of an environment variable and change its behavior accordingly. Different shells support different semantics for exporting environment variables, so see the man page for your preferred shell for further information.

Although child processes of a shell inherit the environment of that shell, shells are separate execution contexts and do not share environment information with one another. Thus, variables you set in one Terminal window are not set in other Terminal windows. Once you close a Terminal window, any variables you set in that window are gone. If you want the value of a variable to persist between sessions and in all Terminal windows, you must set it in a shell startup script.

Another way to set environment variables in Mac OS X is with a special property list in your home directory. At login, the system looks for the following file:

```
~/.MacOSX/environment.plist
```


2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

If the file is present, the system registers the environment variables in the property-list file. For more information on configuring environment variables, see *Runtime Configuration Guidelines*.

Running Programs

To run a program in the shell, you must type the complete pathname of the program's executable file, followed by any arguments, and then press the Return key. If a program is located in one of the shell's known directories, you can omit any path information and just type the program name. The list of known directories is stored in the shell's `PATH` environment variable and includes the directories containing most of the command-line tools.

For example, to run the `ls` command in the current user's home directory, you could simply type it at the command line and press the Return key.

```
host:~ steve$ ls
```

If you wanted to run a tool in the current user's home directory, however, you would need to precede it with the directory specifier. For example, to run the `MyCommandLineProgram` tool, you would use something like the following:

```
host:~ steve$ ./MyCommandLineProgram
```

To launch an application package, you can either use the `open` command (`open MyApp.app`) or launch the application by typing the pathname of the executable file inside the package, usually something like `./MyApp.app/Contents/MacOS/MyApp`.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Mac OS X Frameworks

This appendix contains information about the frameworks of Mac OS X. These frameworks provide the interfaces you need to write software for the platform. Some of these frameworks contain simple sets of interfaces while others contain multiple subframeworks. Where applicable, the tables in this appendix list any key prefixes used by the classes, methods, functions, types, or constants of the framework. You should avoid using any of the specified prefixes in your own symbol names.

System Frameworks

Table B-1 describes the frameworks located in the `/System/Library/Frameworks` directory and lists the first version of Mac OS X in which each became available.

Table B-1 System frameworks

Name	First available	Prefixes	Description
`Accelerate.framework`	10.3	`cblas`, `vDSP`, `vv`	Umbrella framework for vector-optimized operations. See "Accelerate Framework" (page 120).
`AddressBook.framework`	10.2	`AB`, `ABV`	Contains functions for creating and accessing a systemwide database of contact information.
`AGL.framework`	10.0	`AGL`, `GL`, `glm`, `GLM`, `glu`, `GLU`	Contains Carbon interfaces for OpenGL.
`AppKit.framework`	10.0	`NS`	Contains classes and methods for the Cocoa user-interface layer. In general, link to `Cocoa.framework` instead of this framework.
`AppKit-Scripting.framework`	10.0	N/A	Deprecated. Use `AppKit.framework` instead.
`AppleScriptKit.framework`	10.0	`ASK`	Contains interfaces for creating AppleScript plug-ins and provides support for applications built with AppleScript Studio.
`AppleShare-Client.framework`	10.0	`AFP`	Deprecated. Do not use.
`AppleShareClient-Core.framework`	10.0	`AFP`	Contains utilities for handling URLs in AppleShare clients.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Name	First available	Prefixes	Description
AppleTalk.framework	10.0	N/A	Deprecated. Do not use.
ApplicationServices.framework	10.0	AE, AX, ATSU, CG, CT, LS, PM, QD, UT	Umbrella framework for several application-level services. See "Application Services Framework" (page 121).
AudioToolbox.framework	10.0	AU, AUMIDI	Contains interfaces for getting audio stream data, routing audio signals through audio units, converting between audio formats, and playing back music.
AudioUnit.framework	10.0	AU	Contains interfaces for defining Core Audio plug-ins.
Automator.framework	10.4	AM	Umbrella framework for creating Automator plug-ins. See "Automator Framework" (page 121).
CalendarStore.framework	10.5	Cal	Contains interfaces for managing iCal calendar data.
Carbon.framework	10.0	HI, HR, ICA, ICD, Ink, Nav, OSA, PM, SFS,SR	Umbrella framework for Carbon-level services. See "Carbon Framework" (page 122).
Cocoa.framework	10.0	NS	Wrapper for including the Cocoa frameworks AppKit.framework, Foundation.framework, and CoreData.framework.
Collaboration.framework	10.5	CB	Contains interfaces for managing identity information.
CoreAudio.framework	10.0	Audio	Contains the hardware abstraction layer interface for manipulating audio.
CoreAudioKit.framework	10.4	AU	Contains Objective-C interfaces for audio unit custom views.
CoreData.framework	10.4	NS	Contains interfaces for managing your application's data model.
CoreFoundation.framework	10.0	CF	Provides fundamental software services, including abstractions for common data types, string utilities, collection utilities, plug-in support, resource management, preferences, and XML parsing.
CoreMIDI.framework	10.0	MIDI	Contains utilities for implementing MIDI client programs.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Name	First available	Prefixes	Description
CoreMIDIServer.framework	10.0	MIDI	Contains interfaces for creating MIDI drivers to be used by the system.
CoreServices.framework	10.0	CF, DCS, MD, SK, WS	Umbrella framework for system-level services. See "Core Services Framework" (page 123).
CoreVideo.framework	10.5	CV	Contains interfaces for managing video-based content.
Directory-Service.framework	10.0	ds	Contains interfaces for supporting network-based lookup and directory services in your application. You can also use this framework to develop directory service plug-ins.
DiscRecording.framework	10.2	DR	Contains interfaces for burning data to CDs and DVDs.
DiscRecording-UI.framework	10.2	DR	Contains the user interface layer for interacting with users during the burning of CDs and DVDs.
Disk-Arbitration.framework	10.4	DA	Contains interfaces for monitoring and responding to hard disk events.
DrawSprocket.framework	10.0	DSp	Contains the game sprocket component for drawing content to the screen.
DVComponent-Glue.framework	10.0	IDH	Contains interfaces for communicating with digital video devices, such as video cameras.
DVDPlayback.framework	10.3	DVD	Contains interfaces for embedding DVD playback features into your application.
Exception-Handling.framework	10.0	NS	Contains exception-handling classes for Cocoa applications.
ForceFeedback.framework	10.2	FF	Contains interfaces for communicating with force feedback–enabled devices.
Foundation.framework	10.0	NS	Contains the classes and methods for the Cocoa Foundation layer. If you are creating a Cocoa application, linking to the Cocoa framework is preferable.
FWAUserLib.framework	10.2	FWA	Contains interfaces for communicating with FireWire-based audio devices.
GLUT.framework	10.0	glut, GLUT	Contains interfaces for the OpenGL Utility Toolkit, which provides a platform-independent interface for managing windows.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Name	First available	Prefixes	Description
ICADevices.framework	10.3	ICD	Contains low-level interfaces for communicating with digital devices such as scanners and cameras. See also, "Carbon Framework" (page 122).
InputMethodKit.framework	10.5	IMK	Contains interfaces for developing new input methods, which are modules that handle text entry for complex languages.
Installer-Plugins.framework	10.4	IFX	Contains interfaces for creating plug-ins that run during software installation sessions.
InstantMessage.framework	10.4	FZ, IM	Contains interfaces for obtaining the online status of an instant messaging user.
IOBluetooth.framework	10.2	IO	Contains interfaces for communicating with Bluetooth devices.
IOBluetoothUI.framework	10.2	IO	Contains the user interface layer for interacting with users manipulating Bluetooth devices.
IOKit.framework	10.0	IO, IOBSD, IOCF	Contains the main interfaces for developing device drivers.
JavaEmbedding.framework	10.0	N/A	Do not use.
JavaFrame-Embedding.framework	10.5	N/A	Contains interfaces for embedding Java frames in Objective-C code.
JavaScriptCore.framework	10.5	JS	Contains the library and resources for executing JavaScript code within an HTML page. (Prior to Mac OS X v10.5, this framework was part of WebKit.framework.
JavaVM.framework	10.0	JAWT, JDWP, JMM, JNI, JVMDI, JVMPI, JVMTI	Contains the system's Java Development Kit resources.
Kerberos.framework	10.0	GSS, KL, KRB, KRB5	Contains interfaces for using the Kerberos network authentication protocol.
Kernel.framework	10.0	*numerous*	Contains the BSD-level interfaces.
LatentSemantic-Mapping.framework	10.5	LSM	Contains interfaces for classifying text based on latent semantic information.
LDAP.framework	10.0	N/A	Do not use.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Name	First available	Prefixes	Description
Message.framework	10.0	AS, MF, PO, POP, RSS, TOC, UR, URL	Contains Cocoa extensions for mail delivery.
OpenAL.framework	10.4	AL	Contains the interfaces for OpenAL, a cross-platform 3D audio delivery library.
OpenGL.framework	10.0	CGL, GL, glu, GLU	Contains the interfaces for OpenGL, which is a cross-platform 2D and 3D graphics rendering library.
OSAKit.framework	10.4	OSA	Contains Objective-C interfaces for managing and executing OSA-compliant scripts from your Cocoa applications.
PCSC.framework	10.0	MSC, Scard, SCARD	Contains interfaces for interacting with smart card devices.
Preference-Panes.framework	10.0	NS	Contains interfaces for implementing custom modules for the System Preferences application.
PubSub.framework	10.5	PS	Contains interfaces for subscribing to RSS and Atom feeds.
Python.framework	10.3	Py	Contains the open source Python scripting language interfaces.
QTKit.framework	10.4	QT	Contains Objective-C interfaces for manipulating QuickTime content.
Quartz.framework	10.4	GF, PDF, QC, QCP	Umbrella framework for Quartz services. See "Quartz Framework" (page 123)
QuartzCore.framework	10.4	CA,CI, CV	Contains the interfaces for Core Image, Core Animation, and Core Video.
QuickLook.framework	10.5	QL	Contains interfaces for generating thumbnail previews of documents.
QuickTime.framework	10.0	N/A	Contains interfaces for embedding QuickTime multimedia into your application.
Ruby.framework	10.5	N/A	Contains interfaces for the Ruby scripting language.
RubyCocoa.framework	10.5	RB	Contains interfaces for running Ruby scripts from Objective-C code.
ScreenSaver.framework	10.0	N/A	Contains interfaces for writing screen savers.
Scripting.framework	10.0	NS	Deprecated. Use Foundation.framework instead.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Name	First available	Prefixes	Description
Scripting-Bridge.framework	10.5	SB	Contains interfaces for running scripts from Objective-C code.
Security.framework	10.0	CSSM, Sec	Contains interfaces for system-level user authentication and authorization.
Security-Foundation.framework	10.3	Sec	Contains Cocoa interfaces for authorizing users.
Security-Interface.framework	10.3	PSA, SF	Contains the user interface layer for authorizing users in Cocoa applications.
SyncServices.framework	10.4	ISync	Contains the interfaces for synchronizing application data with a central database.
System.framework	10.0	N/A	Do not use.
System-Configuration.framework	10.0	SC	Contains interfaces for accessing system-level configuration information.
Tcl.framework	10.3	Tcl	Contains interfaces for accessing the system's Tcl interpreter from an application.
Tk.framework	10.4	Tk	Contains interfaces for accessing the system's Tk toolbox from an application.
TWAIN.framework	10.2	TW	Contains interfaces for accessing TWAIN-compliant image-scanning hardware.
vecLib.framework	10.0	N/A	Deprecated. Use Accelerate.framework instead. See "Accelerate Framework" (page 120).
WebKit.framework	10.2	DOM, Web	Umbrella framework for rendering HTML content. See "Web Kit Framework" (page 124).
Xgrid-Foundation.framework	10.4	XG	Contains interfaces for connecting to and managing computing cluster software.

Mac OS X contains several umbrella frameworks for major areas of functionality. Umbrella frameworks group several related frameworks into a larger framework that can be included in your project. When writing software, link your project against the umbrella framework; do not try to link directly to any of its subframeworks.The following sections describe the contents of the umbrella frameworks in Mac OS X.

Accelerate Framework

Table B-2 lists the subframeworks of the Accelerate framework (Accelerate.framework). This framework was introduced in Mac OS X version 10.3. If you are developing applications for earlier versions of Mac OS X, vecLib.framework is available as a standalone framework.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Table B-2 Subframeworks of the Accelerate framework

Subframework	Description
`vecLib.framework`	Contains vector-optimized interfaces for performing math, big-number, and DSP calculations, among others.
`vImage.framework`	Contains vector-optimized interfaces for manipulating image data.

Application Services Framework

Table B-3 lists the subframeworks of the Application Services framework (`ApplicationServices.framework`). These frameworks provide C-based interfaces and are intended primarily for Carbon applications, although other programs can use them. The listed frameworks are available in all versions of Mac OS X unless otherwise noted.

Table B-3 Subframeworks of the Application Services framework

Subframework	Description
`ATS.framework`	Contains interfaces for font layout and management using Apple Type Services.
`ColorSync.framework`	Contains interfaces for color matching using ColorSync.
`CoreGraphics.framework`	Contains the Quartz interfaces for creating graphic content and rendering that content to the screen.
`CoreText.framework`	Contains the interfaces for performing text layout and display. Available in Mac OS X v10.5 and later.
`HIServices.framework`	Contains interfaces for accessibility, Internet Config, the pasteboard, the Process Manager, and the Translation Manager. Available in Mac OS X 10.2 and later.
`ImageIO.framework`	Contains interfaces for importing and exporting image data. Prior to Mac OS X v10.5, these interfaces were part of the CoreGraphics subframework.
`LangAnalysis.framework`	Contains the Language Analysis Manager interfaces.
`PrintCore.framework`	Contains the Core Printing Manager interfaces.
`QD.framework`	Contains the QuickDraw interfaces.
`SpeechSynthesis.framework`	Contains the Speech Manager interfaces.

Automator Framework

Table B-4 lists the subframeworks of the Automator framework (`Automator.framework`). This framework was introduced in Mac OS X version 10.4.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Table B-4 Subframeworks of the Automator framework

Subframework	Description
MediaBrowser.framework	Contains private interfaces for managing Automator plug-ins.

Carbon Framework

Table B-5 lists the subframeworks of the Carbon framework (Carbon.framework). The listed frameworks are available in all versions of Mac OS X unless otherwise noted.

Table B-5 Subframeworks of the Carbon framework

Subframework	Description
CarbonSound.framework	Contains the Sound Manager interfaces. Whenever possible, use Core Audio instead.
CommonPanels.framework	Contains interfaces for displaying the Font window, Color window, and some network-related dialogs.
Help.framework	Contains interfaces for launching and searching Apple Help.
HIToolbox.framework	Contains interfaces for the Carbon Event Manager, HIToolbox object, and other user interface–related managers.
HTMLRendering.framework	Contains interfaces for rendering HTML content. For Mac OS X version 10.2 and later, the Web Kit framework is the preferred framework for HTML rendering. See "Web Kit Framework" (page 124).
ImageCapture.framework	Contains interfaces for capturing images from digital cameras. This framework works in conjunction with the Image Capture Devices framework (ICADevices.framework).
Ink.framework	Contains interfaces for managing pen-based input. (Ink events are defined with the Carbon Event Manager.) Available in Mac OS X version 10.3 and later.
Navigation-Services.framework	Contains interfaces for displaying file navigation dialogs.
OpenScripting.framework	Contains interfaces for writing scripting components and interacting with those components to manipulate and execute scripts.
Print.framework	Contains the Carbon Printing Manager interfaces for displaying printing dialogs and extensions.
SecurityHI.framework	Contains interfaces for displaying security-related dialogs.
Speech-Recognition.framework	Contains the Speech Recognition Manager interfaces.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Core Services Framework

Table B-6 lists the subframeworks of the Core Services framework (`CoreServices.framework`). These frameworks provide C-based interfaces and are intended primarily for Carbon applications, although other programs can use them. The listed frameworks are available in all versions of Mac OS X unless otherwise noted.

Table B-6 Subframeworks of the Core Services framework

Subframework	Description
`AE.framework`	Contains interfaces for creating and manipulating Apple events and making applications scriptable.
`CarbonCore.framework`	Contains interfaces for many legacy Carbon Managers. In Mac OS X v10.5 and later, this subframework contains the FSEvents API, which notifies clients about file system changes.
`CFNetwork.framework`	Contains interfaces for network communication using HTTP, sockets, and Bonjour.
`DictionaryServices.framework`	Provides dictionary lookup capabilities.
`LaunchServices.framework`	Contains interfaces for launching applications.
`Metadata.framework`	Contains interfaces for managing Spotlight metadata. Available in Mac OS X v10.4 and later.
`OSServices.framework`	Contains interfaces for Open Transport and many hardware-related legacy Carbon managers.
`SearchKit.framework`	Contains interfaces for the Search Kit. Available in Mac OS X version 10.3 and later.

Quartz Framework

Table B-7 lists the subframeworks of the Quartz framework (`Quartz.framework`). This framework was introduced in Mac OS X version 10.4.

Table B-7 Subframeworks of the Quartz framework

Subframework	Description
`ImageKit.framework`	Contains Objective-C interfaces for finding, browsing, and displaying images. Available in Mac OS X version 10.5 and later.
`PDFKit.framework`	Contains Objective-C interfaces for displaying and managing PDF content in windows.
`QuartzComposer.framework`	Contains Objective-C interfaces for playing Quartz Composer compositions in an application.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Subframework	Description
QuartzFilters.framework	Contains Objective-C interfaces for managing and applying filter effects to a graphics context. Available in Mac OS X version 10.5 and later.

Web Kit Framework

Table B-8 lists the subframeworks of the Web Kit framework (WebKit.framework). This framework was introduced in Mac OS X version 10.2.

Table B-8 Subframeworks of the Web Kit framework

Subframework	Description
WebCore.framework	Contains the library and resources for rendering HTML content in an HTMLView control.

Xcode Frameworks

In Mac OS X v10.5 and later, Xcode and all of its supporting tools and libraries reside in a portable directory structure. This directory structure makes it possible to have multiple versions of Xcode installed on a single system or to have Xcode installed on a portable hard drive that you plug in to your computer when you need to do development. This portability means that the frameworks required by the developer tools are installed in the <Xcode>/Library/Frameworks directory, where <Xcode> is the path to the Xcode installation directory. (The default Xcode installation directory is /Developer.) Table B-9 lists the frameworks that are located in this directory.

Table B-9 Xcode frameworks

Framework	First available	Prefixes	Description
CPlusTest.framework	10.4	None	Unit-testing framework for C++ code. In Mac OS X v10.4, this framework was in /System/Library/Frameworks.
InterfaceBuilder-Kit.framework	10.5	ib, IB	Contains interfaces for writing plug-ins that work in Interface Builder v3.0 and later.
SenTesting-Kit.framework	10.4	Sen	Contains the interfaces for implementing unit tests in Objective-C. In Mac OS X v10.4, this framework was in /System/Library/Frameworks.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

System Libraries

Note that some specialty libraries at the BSD level are not packaged as frameworks. Instead, Mac OS X includes many dynamic libraries in the /usr/lib directory and its subdirectories. Dynamic shared libraries are identified by their .dylib extension. Header files for the libraries are located in /usr/include.

Mac OS X uses symbolic links to point to the most current version of most libraries. When linking to a dynamic shared library, use the symbolic link instead of a link to a specific version of the library. Library versions may change in future versions of Mac OS X. If your software is linked to a specific version, that version might not always be available on the user's system.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Mac OS X Developer Tools

Apple provides a number of applications and command-line tools to help you develop your software. These tools include compilers, debuggers, performance analysis tools, visual design tools, scripting tools, version control tools, and many others. Many of these tools are installed with Mac OS X by default but the rest require you to install Xcode first. Xcode is available for free from the Apple Developer Connection website. For more information on how to get these tools, see "Getting the Xcode Tools" (page 14).

> **Note:** Documentation for most of the command-line tools is available in the form of `man` pages. You can access these pages from the command line or from *Mac OS X Man Pages*. For more information about using the command-line tools, see "Command Line Primer" (page 109).

Applications

Xcode includes numerous applications for writing code, creating resources, tuning your application, and delivering it to customers. At the heart of this group is the Xcode application, which most developers use on a daily basis. It provides the basic project and code management facilities used to create most types of software on Mac OS X. All of the tools are free and can be downloaded from the Apple developer website (see "Getting the Xcode Tools" (page 14)).

In Mac OS X v10.5 and later, it is possible to install multiple versions of Xcode on a single computer and run the applications and tools from different versions side-by-side. The applications listed in the following sections are installed in <*Xcode*>`/Applications`, where <*Xcode*> is the root directory of your Xcode installation. The default installation directory for Xcode is the `/Developer` directory.

In addition to the applications listed here, Xcode also comes with numerous command-line tools. These tools include the GCC compiler GDB debugger, tuning tools, code management tools, performance tools, and so on. For more information about the available command-line tools, see "Command-Line Tools" (page 143).

Xcode

The centerpiece of the Xcode Tools is the Xcode application, which is an integrated developer environment (IDE) with the following features:

- A project management system for defining software products
- A code editing environment that includes features such as syntax coloring, code completion, and symbol indexing; see"Xcode Editor" (page 129)
- Visual design tools for creating your application's data model (see "Core Data Framework" (page 60))
- An advanced documentation viewer for viewing and searching Apple documentation; see "Documentation Window" (page 130)

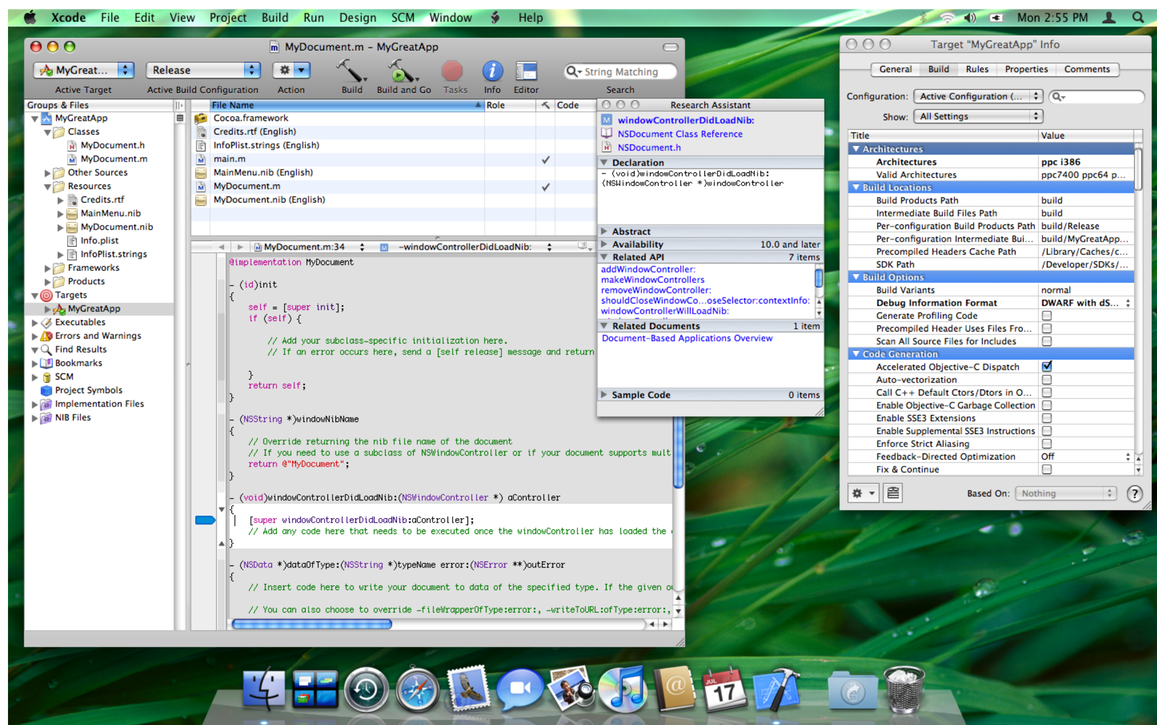
2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

- A context-sensitive inspector for viewing information about selected code symbols; see "Research Assistant" (page 130)
- An advanced build system with dependency checking and build rule evaluation.
- GCC compilers supporting C, C++, Objective-C, Objective-C++, Objective-C 2.0, and other compilers supporting Java and other languages
- Integrated source-level debugging using GDB; see "Debugging Environment" (page 130)
- Distributed computing, enabling you to distribute large projects over several networked machines
- Predictive compilation that speeds single-file compile turnaround times
- Advanced debugging features such as fix and continue and custom data formatters
- Advanced refactoring tools that let you make global modifications to your code without changing its overall behavior; see "Refactoring Tools" (page 132)
- Support for project snapshots, which provide a lightweight form of local source code management; see "Project Snapshots" (page 132)
- Support for launching performance tools to analyze your software
- Support for integrated source-code management; see "SCM Repository Management" (page 131)
- AppleScript support for automating the build process
- Support for the ANT build system, which can be used to build Java and WebObjects projects.
- Support for DWARF and Stabs debugging information (DWARF debugging information is generated by default for all new projects)

Figure C-1 shows the Xcode project workspace and some key inspector windows. In the Xcode preferences, you can configure numerous aspects of the workspace to suit your preferred work style.

© 2004, 2008 Apple Inc. All Rights Reserved.

Figure C-1 Xcode application



For information on how to use Xcode, see *Xcode User Guide*.

Xcode Editor

The Xcode editing environment is a high-performance code editor that includes many features that go beyond basic text editing. These features aim to help developers create better code faster and include the following:

- High-performance for typing, scrolling, and opening files. The Xcode editor now opens and scrolls large source documents up to 10 times faster than before.
- Code annotations display notes, errors, and warnings inline with the code itself, and not just as icons in the gutter. This provides a much more direct conveyance of where the problems in your code lie. You can control the visibility of annotations using the segmented control in the navigation bar.
- Code folding helps you organize your source files by letting you temporarily hide the content of a method or function in the editor window. You can initiate code folding by holding down the Command and Option keys and pressing either the left or right arrow key. A ribbon to the left of the text shows the current nesting depth and contains widgets to fold and unfold code blocks.
- Syntax coloring lets you assign colors to various code elements, including keywords, comments, variables, strings, class names, and more.
- Code Sense code completion, a feature that shows you type a few characters and retrieve a list of valid symbol names that match those characters. Code Sense is fast and intuitive and is tuned to provide accurate completions, along with a "most likely" inline completion as you type. This feature is similar to the auto-completion features found in Mail, Terminal, and other applications.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Debugging Environment

In Xcode 3.0, there is no distinction between "Running" your executable and "Debugging" it. Instead, you simply build your executable and run it. Hitting a breakpoint interrupts the program and displays the breakpoint either in the current editor window or in the debugger window. Other features of the debugging environment include the following:

- Debugging controls in editor windows.
- A debugger HUD (heads-up-display), which is a floating window with debugger controls that simplifies the debugging of full-screen applications.
- Variable tooltips. (Moving your mouse over any variable displays that variable's value.)
- Reorganization (and in some cases consolidation) of toolbar and menu items to improve space usage, while still keeping all the needed tools available.
- Consolidation of the Standard I/O Log, Run Log, and Console log into the Console log window.
- Support for a separate debugging window if you prefer to debug your code that way.

Research Assistant

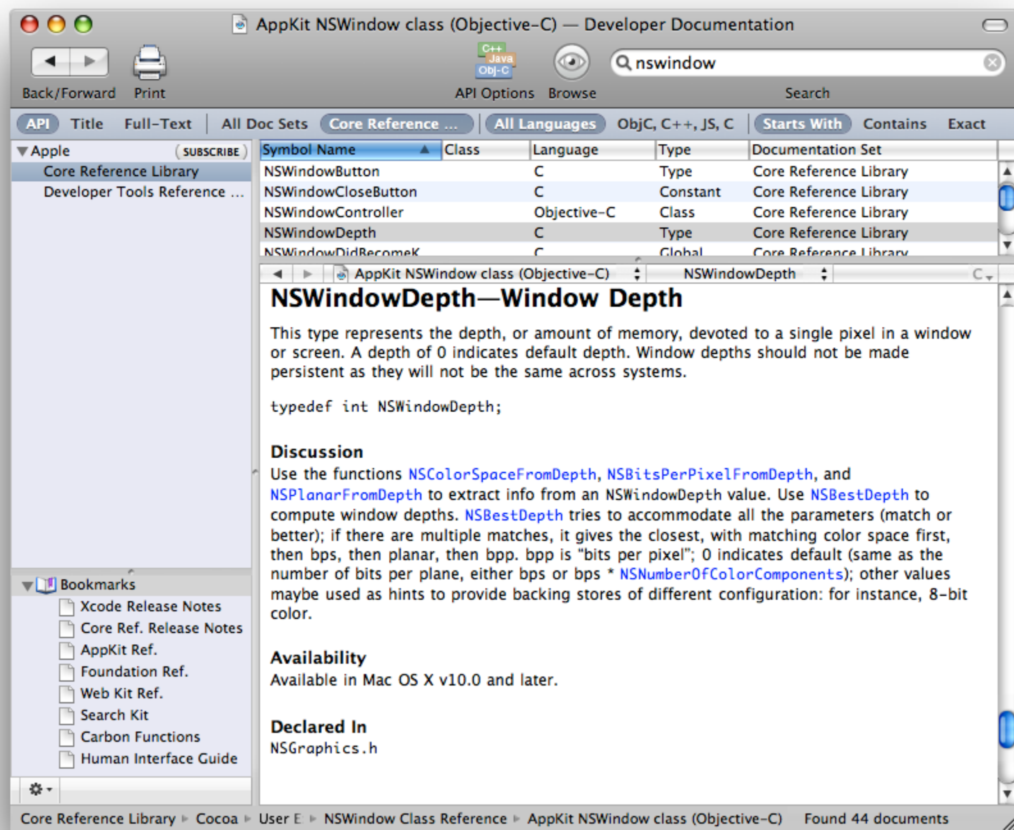
The Research Assistant is an inspector that displays documentation for the currently selected text (see Figure C-1 (page 129)). As the selection changes, the Research Assistant updates the information in its floating window to reflect the classes, methods, and functions you are currently using. This window shows the declaration, abstract, and availability information for the selection along with the framework containing the selected identifier, relevant documentation resources, and related methods and functions you might be interested in using.

Documentation Window

The documentation window (Figure C-2) in Xcode provides an environment for searching and browsing the documentation. This window provides you with fast access to Apple's developer documentation and gives you tools for searching its content. You can search by title, by language, and by content and can focus your search on the documents in a particular documentation set.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Figure C-2 Xcode documentation window



Documentation sets are collections of documents that can be installed, browsed, and searched independently. Documentation sets make it easier to install only the documentation you need for your development, reducing the amount of disk space needed for your developer tools installation. In addition to the Apple-provided documentation sets, third parties can implement their own documentation sets and have them appear in the Xcode documentation window. For information on how to create custom documentation sets, see *Documentation Set Guide*.

SCM Repository Management

Xcode supports the management of multiple SCM repositories to allow you to perform tasks such as the following:

- Initial checkout of projects
- Tagging source files
- Branching
- Importing and exporting files

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Xcode supports CVS, Subversion, and Perforce repositories.

Project Snapshots

Project snapshots provide a lightweight form of local source control for Xcode projects. Using this feature, you can take a "snapshot" of your project's state at any point during development, such as after a successful build or immediately prior to refactoring your code. If after making subsequent changes you decide those changes were not useful, you can revert your project files back to the previous snapshot state. Because snapshots are local, your intermediate changes need never be committed to source control.

Refactoring Tools

Xcode's refactoring tools let you make large-scale changes to your Objective-C source code quickly and easily. Xcode propagates your change requests throughout your code base, making sure that the changes do not break your builds. You can make the following types of changes using the refactoring tools:

- Rename instance methods
- Create new superclasses
- Move methods into a superclass
- Convert accessor methods to support Objective-C 2.0 properties
- Modernize appropriate `for` loops to use the new fast enumeration syntax introduced in Objective-C 2.0

Before making any changes to your code, Xcode's refactoring tools automatically take a local snapshot of your project. This automatic snapshot means you can experiment with refactoring changes without worrying about irrevocably changing your project files. For more information on snapshots, see "Project Snapshots" (page 132).

Build Settings

The Build pane in the inspector organizes the build settings for the selected target, providing search tools to help you find particular settings. In Mac OS X v10.5, some particularly noteworthy additions to this pane include the following:

- Per-architecture build settings. You can now set different build settings for each architecture (Intel, PowerPC) your product supports.
- 32-bit and 64-bit architecture checkboxes.

Project Versioning

Xcode projects include a Compatibility pane in the project inspector that lets you determine whether you want an Xcode 3.0–only project or one that can be used by previous versions of Xcode. Marking a project as Xcode 3.0–only generates an alert whenever you try to use an Xcode feature that is not present in previous versions of the application.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

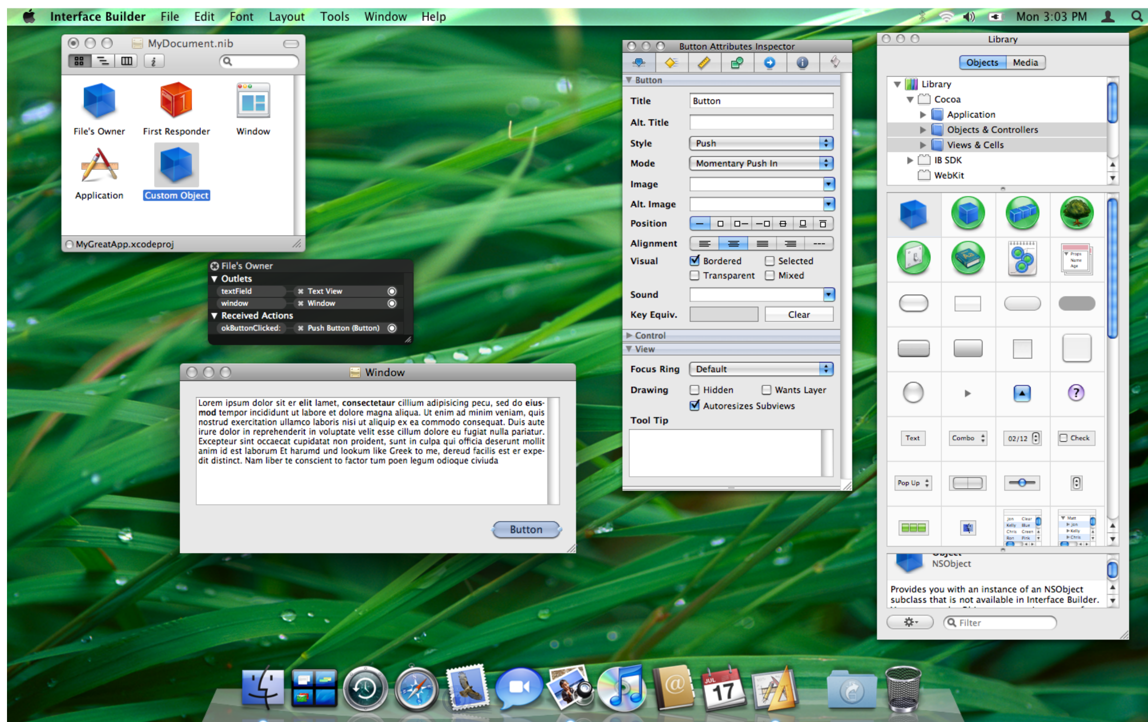
Interface Builder

The Interface Builder application provides a graphical environment for building the user interface of your Carbon and Cocoa applications. Using Interface Builder, you assemble the windows and menus of your application, along with the any other supporting objects, and save them in one or more resource files, called nib files. When you want to load a user interface element at runtime, you load the nib file. The Cocoa and Carbon infrastructure uses the data in the nib file to recreate the objects exactly as they were in Interface Builder, with all their attributes and inter-object relationships restored.

Although present in all versions of Mac OS X, the Interface Builder application received a significant overhaul in Mac OS X v10.5. Beyond the numerous cosmetic changes, the current version of Interface Builder includes numerous workflow and infrastructure changes too. The connections panel replaces the old technique for connecting objects in Cocoa nib files, making it possible to create multiple connections quickly without going back and forth between the inspector and the objects in your nib file. An improved library window helps you organize and find the components you use most frequently. Interface Builder includes a new plug-in model that makes it possible to create fully functional plug-ins in a matter of minutes. And most importantly, Interface Builder is more tightly integrated with Xcode, providing automatic synchronization of project's class information with the corresponding source files.

Figure C-3 shows the Interface Builder environment in Mac OS X v10.5, including a nib document, connections panel, inspector window, and library window. The library window contains the standard components you use to build your user interfaces and includes all of the standard controls found in Carbon and Cocoa applications by default. Using plug-ins, you can expand the library to include your own custom objects or to include custom configurations of standard controls.

Figure C-3 Interface Builder 3.0



2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

For information about Interface Builder features and how to use them, see *Interface Builder User Guide*. For information about how to integrate your own custom controls into Interface Builder, see *Interface Builder Plug-In Programming Guide* and *Interface Builder Kit Framework Reference*.

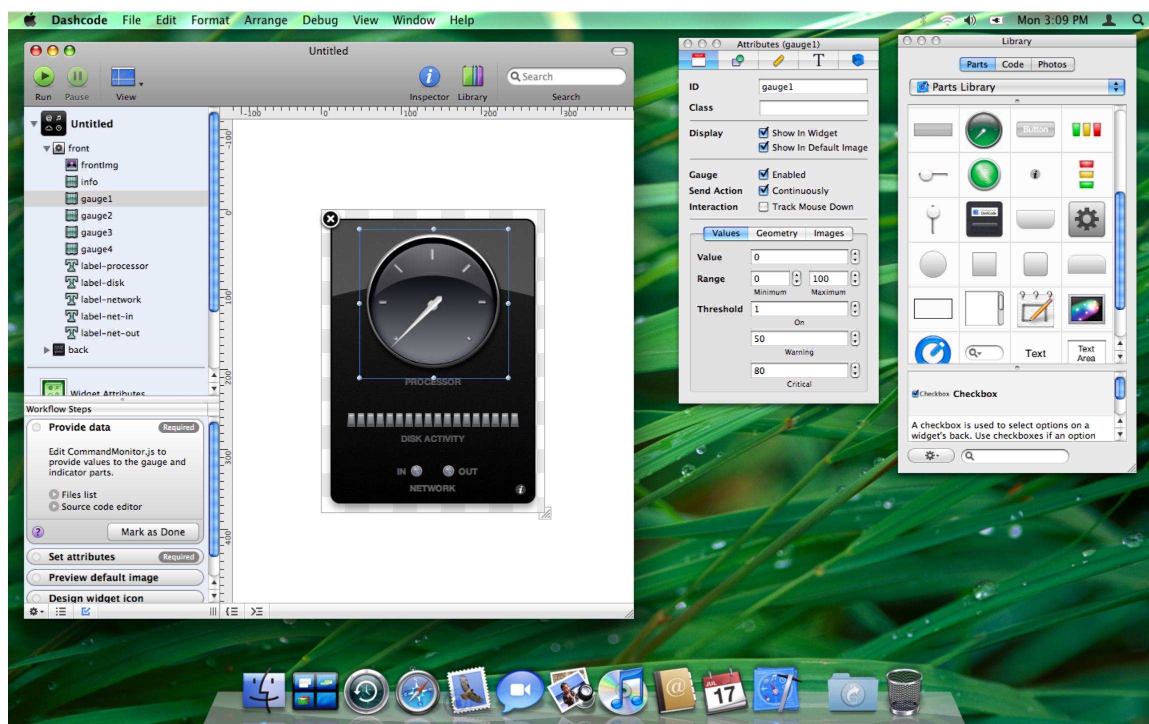
Dashcode

Introduced in Mac OS X v10.5, Dashcode is an integrated environment for laying out, coding, and testing Dashboard widgets. Although users see and use widgets as applications, they're actually packaged webpages powered by standard technologies such as HTML, CSS, and JavaScript. Although it is easy for anyone with web design experience to build a widget using existing webpage editors, as a widget's code and layout get more complex, managing and testing of that widget becomes increasingly difficult. Dashcode provides the following features to help simplify the widget design process:

- A project manager to marshall your widget's resources
- Visual tools to design your widget interface
- Tools to set metadata values, specify required images, and package your widget
- A source code editor to implement your widget's behavior
- A debugger to help you resolve issues in your widget's implementation

Figure C-4 shows the Dashcode canvas, inspector, and library windows. The canvas is a drag-and-drop layout environment where you lay out widgets visually. Using the inspector window, you can apply style information to the controls, text, and shape elements that you drag in from the library.

Figure C-4 Dashcode canvas



2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

For more information about Dashcode, see *Dashcode User Guide*.

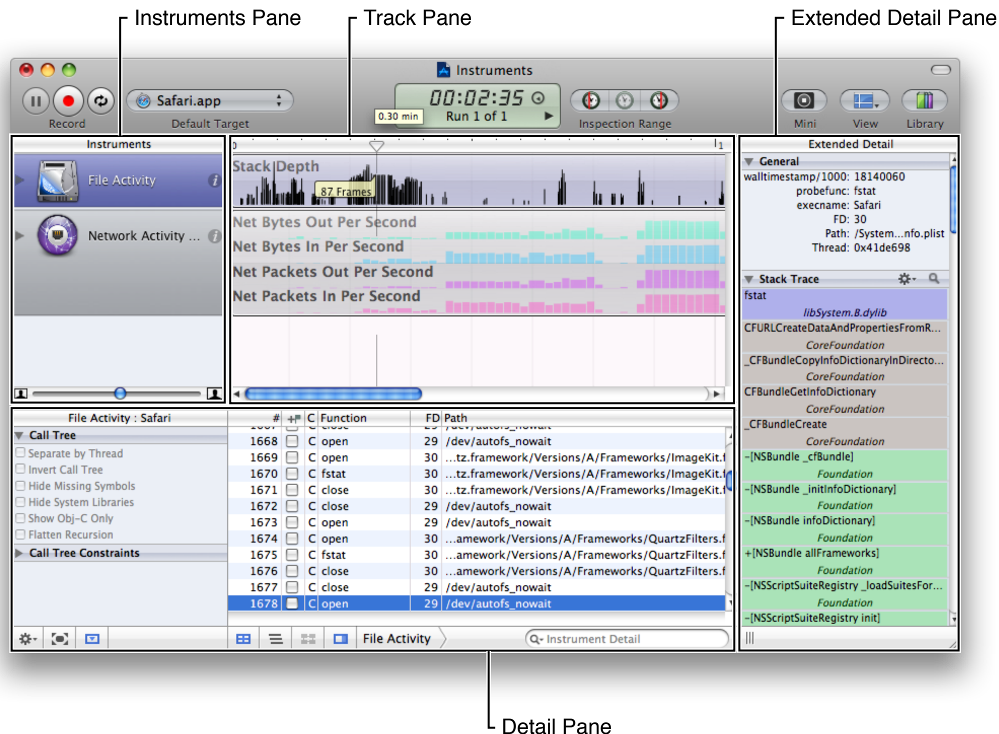
Instruments

Introduced in Mac OS X v10.5, Instruments is an advanced debugging and performance analysis application. Instruments provides unprecedented information about the runtime behavior of your application and complements existing tools such as Shark. Rather than show one aspect of your program at a time, you configure each analysis session with one or more "instruments", each of which gathers information about things such as object allocation patterns, memory usage, disk I/O, CPU usage, and many more. The data from all instruments is shown side-by-side, making it easier to see patterns between different types of information.

An important aspect of Instruments is the repeatability of data gathering operations. Instruments lets you record a sequence of events in your application and store them in the master track. You can then replay that sequence to reproduce the exact same conditions in your application. This repeatability means that each new set of data you gather can be compared directly to any old sets, resulting in a more meaningful comparison of performance data. It also means that you can automate much of the data gathering operation. Because events are shown alongside data results, it is easier to correlate performance problems with the events that caused them.

Figure C-5 shows the Instruments user interface for an existing session. Data for each instrument is displayed along the horizontal axis. Clicking in those data sets shows you information about the state of the application at that point in time.

Figure C-5 The Instruments application interface



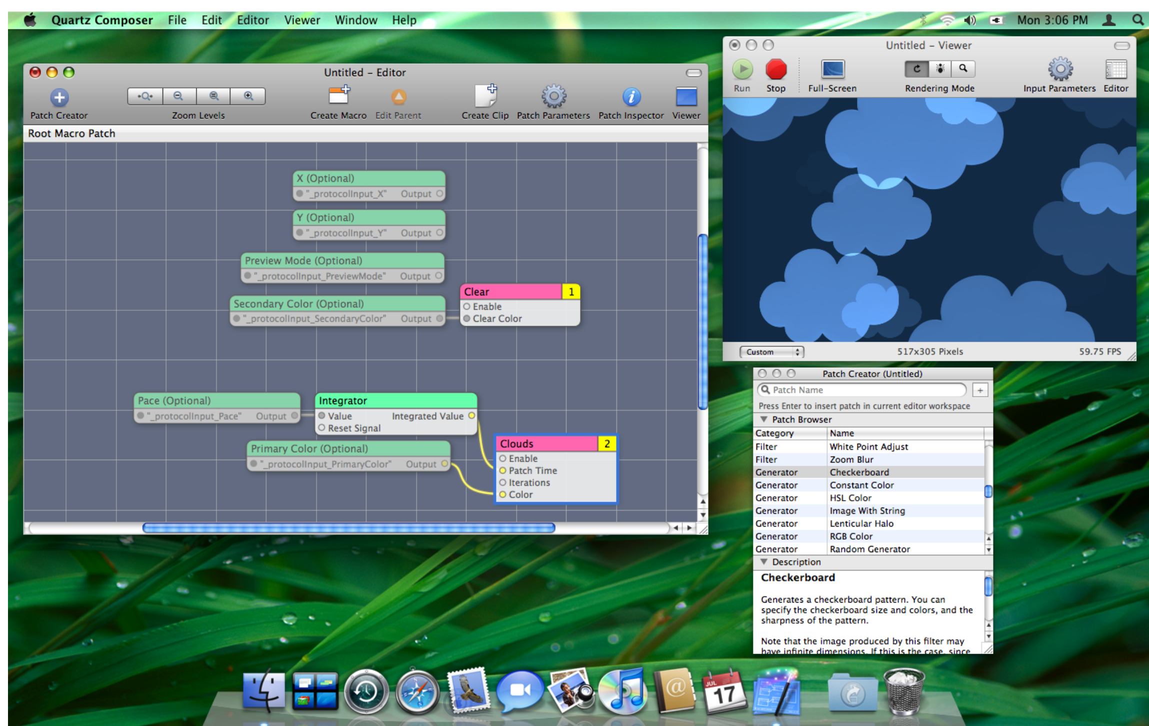
For information about how to use Instruments, see *Instruments User Guide*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Quartz Composer

Introduced in Mac OS X version 10.4, Quartz Composer is a development tool for processing and rendering graphical data. Quartz Composer provides a visual development environment (Figure C-6) built on technologies such as Quartz 2D, Core Image, OpenGL, and QuickTime. You can use Quartz Composer as an exploratory tool to learn the tasks common to each visual technology without having to learn its application programming interface (API). You can also save your compositions as resource files that can be loaded into a Cocoa window at runtime. In addition to supporting visual technologies, Quartz Composer also supports nongraphical technologies such as MIDI System Services and Rich Site Summary (RSS) file content.

Figure C-6 Quartz Composer editor window



For information on how to use Quartz composer, see *Quartz Composer User Guide*.

AppleScript Studio

You can use AppleScript Studio to create AppleScript applications with complex user interfaces that support the Aqua human interface guidelines. AppleScript Studio is a combination of technologies, including AppleScript, Cocoa, the Xcode application, and Interface Builder.

For more information about AppleScript Studio, see *AppleScript Studio Programming Guide* or *AppleScript Studio Terminology Reference*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Audio Applications

The <*Xcode*>`/Applications/Audio` directory contains applications for Core Audio developers.

AU Lab

Introduced in Mac OS X version 10.4, AU Lab (Audio Unit Lab) lets you graphically host audio units and examine the results. You can use AU Lab to test the audio units you develop, do live mixing, and playback audio content. Parameters for the audio units are controlled graphically using the audio unit's custom interface or using a generic interface derived from the audio unit definition. Figure C-7 shows the AU Lab interface and some of the palettes for adjusting the audio parameters.

Figure C-7 AU Lab mixer and palettes

HALLab

Introduced in Mac OS X version 10.5, the HALLab (Hardware Abstraction Layer Lab) application helps developers test and debug audio hardware and drivers. You can use this application to understand what the audio hardware is doing and to correlate the behavior of your application with the behavior of the underlying audio driver. The application provides information about the properties of objects in the HAL and provides an I/O cycle telemetry viewer for diagnosing and debugging glitches your application's audio content.

Graphics Applications

Table C-1 lists the applications found in the <*Xcode*>`/Applications/Graphics Tools` directory.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Table C-1 Graphics applications

Application	Description
Core Image Fun House	Introduced in Mac OS X v10.5, this application provides an environment for testing the effects of Core Image filters. Using this application, you can build up a set of filters and apply them to an image or set of images. You can apply both static and time-based effects and change the parameters of each filter dynamically to see the results.
OpenGL Driver Monitor	An application that displays extensive information about the OpenGL environment.
OpenGL Profiler	An application that creates a runtime profile of an OpenGL-based application. The profile contains OpenGL function-call timing information, a listing of all the OpenGL function calls your application made, and all the OpenGL-related data needed to replay your profiling session.
OpenGL Shader Builder	An application that provides real-time entry, syntax checking, debugging, and analysis of vertex/fragment programs. It allows you to export your creation to a sample GLUT application, which performs all the necessary OpenGL setup, giving you a foundation to continue your application development. OpenGL is an open, cross-platform, three-dimensional (3D) graphics standard that supports the abstraction of current and future hardware accelerators. For more information about OpenGL, see *OpenGL Programming Guide for Mac OS X* in the Reference Library > Graphics & Imaging area.
Pixie	A magnifying glass utility for Mac OS X. Pixie is useful for doing pixel-perfect layout, checking the correctness of graphics and user interface elements, and getting magnified screen shots.
Quartz Composer Visualizer	A utility for previewing Quartz Composer compositions.
Quartz Debug	This is an alias to the Quartz Debug application in the <*Xcode*>`/Applications/Performance Tools` directory. For more information, see the entry for Quartz Debug in "Performance Applications" (page 139).

Java

Table C-2 lists the applications found in the <*Xcode*>`/Applications/Java Tools` directory.

Table C-2 Java applications

Application	Description
Applet Launcher	An application that acts as a wrapper for running Java applets.
Jar Bundler	An application that allows you to package your Java program's files and resources into a single double-clickable application bundle. Jar Bundler lets you modify certain properties so your Java application behaves as a better Mac OS X citizen and lets you specify arguments sent to the Java virtual machine (VM) when the application starts up.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Performance Applications

Table C-3 lists the applications found in the <*Xcode*>`/Applications/Performance Tools` directory.

Table C-3 Performance applications

Application	Description
BigTop	An application that presents statistics about the current system activity and lets you track those statistics over time. This application is a more visually oriented version of the top command-line tool. It provides information about CPU usage, disk and network throughput, memory usage, and others. For information on how to use this program, see the application help.
MallocDebug	An application for measuring the dynamic memory usage of applications and for finding memory leaks. For information on how to use this program, see the application help or *Memory Usage Performance Guidelines*.
Quartz Debug	A debugging utility for the Quartz graphics system. For information on how to use this program, see the application help or *Drawing Performance Guidelines*.
Shark	An application that profiles the system to see how time is being spent. It can work at the system, task, or thread level and can correlate performance counter events with source code. Shark's histogram view can be used to observe scheduling and other time-dependent behavior. It can produce profiles of hardware and software performance events such as cache misses, virtual memory activity, instruction dependency stalls, and so forth. For information on how to use this program, see the application help.
Spin Control	An application that samples applications automatically whenever they become unresponsive and display the spinning cursor. To use this application, you launch it and leave it running. Spin Control provides basic backtrace information while an application is unresponsive, showing you what the application was doing at the time.
Thread Viewer	An application for graphically displaying activity across a range of threads. It provides timeline color-coded views of activity on each thread. By clicking a point on a timeline, you can see a sample backtrace of activity at that time.
ZoneMonitor	An application for analyzing memory usage.

Table C-4 lists the applications in the <*Xcode*>`/Applications/Performance Tools/CHUD` directory and its subdirectories.

Table C-4 CHUD applications

Application	Description
Reggie SE	An application that examines and modifies CPU and PCI configuration registers in PowerPC processors.
PMC Index	An application for finding performance counter events and their configuration.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Application	Description
Saturn	An application that is an exact, function-level profiler for your application. Unlike sampling programs, which gather call stacks at periodic intervals, you can use this application to generate and view a complete function call trace of your application code.
SpindownHD	An application that monitors the power state of hard drives connected to the computer.

Utility Applications

Table C-5 lists the applications found in the <*Xcode*>`/Applications/Graphics Tools` directory and its subdirectories.

Table C-5 Utility applications

Application	Description
Accessibility Inspector	An agent application that lets you roll the mouse cursor over items in your application's user interface and view their associated accessibility attributes and actions.
Accessibility Verifier	An application that looks for mistakes in the accessibility information provided by your application.
Bluetooth Explorer	An application for discovering and getting information about Bluetooth devices.
Build Applet	An application for creating applets from Python scripts.
Clipboard Viewer	An application that displays the contents of the various system pasteboards.
CrashReporterPrefs	An application for configuring the user notifications generated when an application crashes.
FileMerge	An application that compares two ASCII files or two directories. For a more accurate comparison, you can compare two files or directories to a common ancestor. After comparing, you can merge the files or directories.
Help Indexer	An application to create a search index for a help file. Instructions for creating Apple Help and for using the indexing tool are in *Apple Help Programming Guide*.
Icon Composer	An application for creating and examining icon resource files.
IORegistryExplorer	An application that you can use to examine the configuration of devices on your computer. IORegistryExplorer provides a graphical representation of the I/O Registry tree. For information on how to use this application, see *I/O Kit Fundamentals*.
iSync Plug-in Maker	This application creates device drivers that allow the synchronization of custom hardware devices. For more information, see "iSync Plug-in Maker" (page 141).
PackageMaker	An application for creating installable application packages from the set of files you provide.
PacketLogger	An application for logging Bluetooth packets.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Application	Description
Property List Editor	An application that lets you read and edit the contents of a property list. A property list, or plist, is a data representation used by Cocoa and Core Foundation as a convenient way to store, organize, and access standard object types. Property lists are useful when you need to store small amounts of persistent data. If you do use property lists, the `.plist` files must be bundled with other application files when you create your installation package.
Repeat After Me	An application designed to improve the pronunciation of text generated by the Text-To-Speech system.
SRLanguageModeler	An application for building language models for use with the Speech Recognition manager.
Syncrospector	A debugging application you use to inspect the truth database, the call history of sync sessions, and clients of the synchronization engine. For information on how to use this application, see *Sync Services Tutorial*.
USB Prober	An application that displays detailed information about all the USB ports and devices on the system.

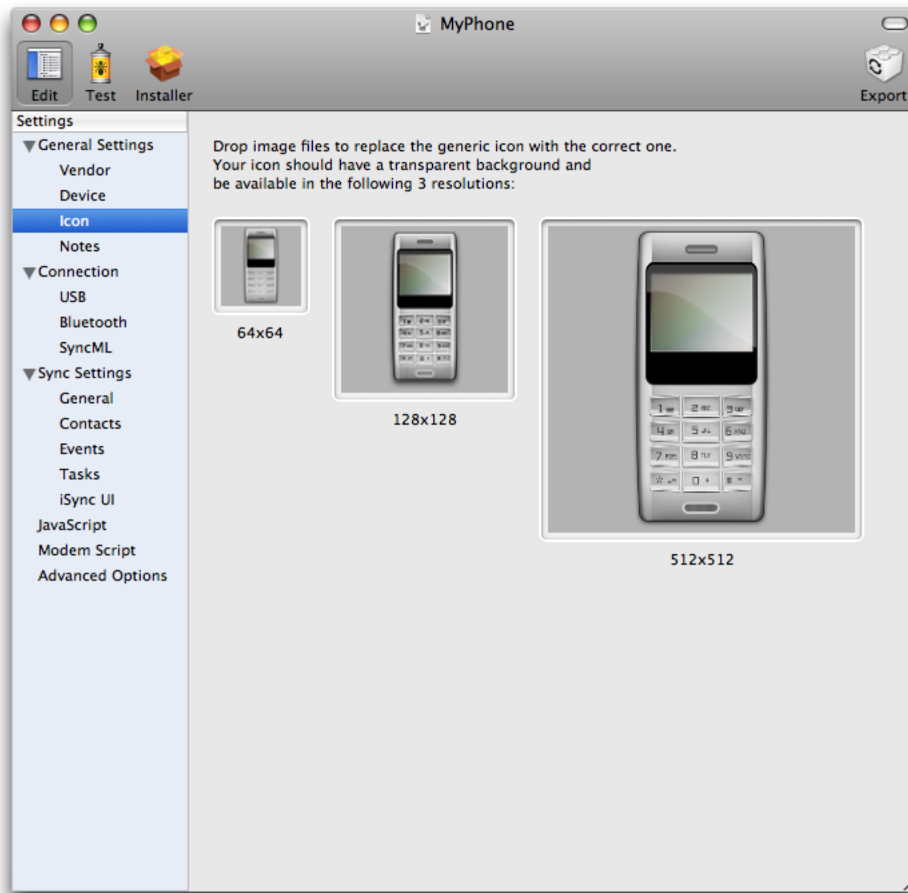
iSync Plug-in Maker

The iSync Plug-in Maker application is a tool that allows you to build, test, and release plug-ins that handle the specific features supported by your hardware device. You use this application to configure your device settings and write scripts for connecting it to the Internet. The application also provides a suite of standard automated tests that you can use to detect and fix problems in your plug-in before you ship it.

Figure C-8 shows the iSync Plug-in Maker edit window.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Figure C-8 iSync Plug-in Maker application



For information about using iSync Plug-in Maker, see *iSync Plug-in Maker User Guide*.

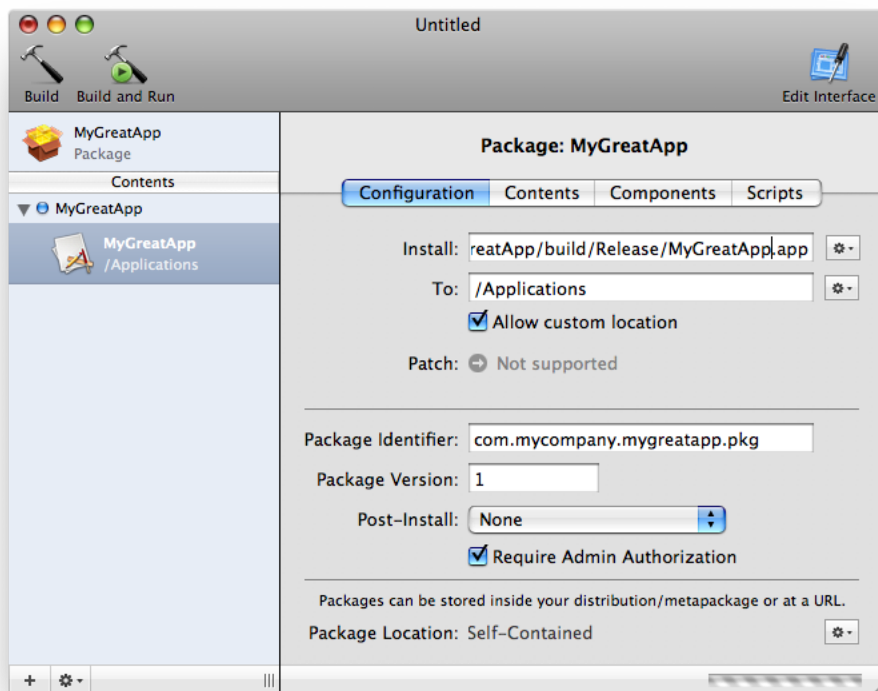
PackageMaker

You use PackageMaker to create installation packages for your software. An installation package is a convenient way to deliver software in all but the simplest cases. An installation package contains the files to install, their locations, and any licensing information or supporting materials that go with your software. When the user double-clicks an installation package, Mac OS X automatically launches the Installer application, which proceeds to install the files contained in the package.

You can use PackageMaker to package files or to assemble individual packages into a single package. Figure C-9 shows the PackageMaker user interface, which provides a graphical environment for building your packages.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Figure C-9 PackageMaker application



For information on how to use PackageMaker, see *PackageMaker User Guide*.

Command-Line Tools

Xcode includes numerous command-line tools, including the GCC compiler, GDB debugger, performance tools, version control system tools, localization tools, scripting tools, and so on. Some of these tools are found on most other BSD-based installations while others were created specifically for Mac OS X. They come free with the rest of Xcode, which you can download from the Apple developer website (see "Getting the Xcode Tools" (page 14)).

In Mac OS X v10.5 and later, it is possible to install multiple versions of Xcode on a single computer and run the applications and tools from different versions side-by-side. Most of the tools listed in the following sections are installed in either in the system's `/usr/bin` directory or in <*Xcode*>`/usr/bin` or <*Xcode*>`/usr/sbin`, where <*Xcode*> is the root directory of your Xcode installation, although tools installed elsewhere are called out as such. The default installation directory for Xcode is the `/Developer` directory.

In addition to the command-line tools listed here, Xcode also comes with many higher-level applications. These tools include the Xcode integrated development environment, Interface Builder, Instruments, and many others. For more information about the available applications, see "Applications" (page 127).

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Note: The following sections describe some of the more important tools provided with Xcode but should by no means be considered a complete list. If the tool you are looking for is not described here, check the system and Xcode tools directories or see *Mac OS X Man Pages*.

Compiler, Linker, and Source Code Tools

Apple provides several applications and command-line tools for creating source code files.

Compilers, Linkers, Build Tools

Table C-6 lists the command-line compilers, linkers, and build tools. These tools are located in <*Xcode*>`/usr/bin` and <*Xcode*>`/Private`.

Table C-6 Compilers, linkers, and build tools

Tool	Description
`as`	The Mac OS X Mach-O assembler. See `as` man page.
`bsdmake`	The BSD make program. See `bsdmake` man page.
`gcc`	The command-line interface to the GNU C compiler (GCC). Normally you invoke GCC through the Xcode application; however, you can execute it from a command line if you prefer. See `gcc` man page.
`gnumake`	The GNU make program. See `gnumake` man page.
`jam`	An open-source build system initially released by Perforce, which provides the back-end for the Xcode application's build system. It is rarely used directly from the command line. Documented on the Perforce website at http://www.perforce.com/jam/jam.html.
`ld`	Combines several Mach-O (Mach object) files into one by combining like sections in like segments from all the object files, resolving external references, and searching libraries. Mach-O is the native executable format in Mac OS X. See `ld` man page.
`make`	A symbolic link to `gnumake`, the GNU make program. Note that the Xcode application automatically creates and executes make files for you; however the command-line make tools are available if you wish to use them. See `make` man page.
`mkdep`	Constructs a set of include file dependencies. You can use this command in a make file if you are constructing make files instead of using Xcode to build and compile your program. See `mkdep` man page.
`pbprojectdump`	Takes an Xcode project (`.pbproj`) file and outputs a more nested version of the project structure. Note that, due to how conflicts are reflected in the project file, `pbprojectdump` cannot work with project files that have CVS conflicts.
`xcodebuild`	Builds a target contained in an Xcode project. This command is useful if you need to build a project on another computer that you can connect to with Telnet. The `xcodebuild` tool reads your project file and builds it just as if you had used the Build command from within the Xcode application. See `xcodebuild` man page.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Library Utilities

Table C-7 lists the command-line tools available for creating libraries. These tools are located in *<Xcode>*`/usr/bin`.

Table C-7 Tools for creating and updating libraries

Tool	Description
`libtool`	Takes object files and creates dynamically linked libraries or archive (statically linked) libraries, according to the options selected. The `libtool` command calls the `ld` command. See `libtool` man page.
`lorder`	Determines interdependencies in a list of object files. The output is normally used to determine the optimum ordering of the object modules when a library is created so that all references can be resolved in a single pass of the loader. See `lorder` man page.
`ranlib`	Adds to or updates the table of contents of an archive library. See `ranlib` man page.
`redo_prebinding`	Updates the prebinding of an executable or dynamic library when one of the dependent dynamic libraries changes. (Prebinding for user applications is unnecessary in Mac OS X v10.3.4 and later.) See `redo_prebinding` man page.
`update_prebinding`	Updates prebinding information for libraries and executables when new files are added to the system. (Prebinding for user applications is unnecessary in Mac OS X v10.3.4 and later.) See `update_prebinding` man page.

Code Utilities

Table C-8 lists applications and command-line tools for manipulating source code and application resources. These tools are located in *<Xcode>*`/usr/bin` and `/usr/bin`.

Table C-8 Code utilities

Tool	Description
`fpr`	Reformats a Fortran file for printing by the UNIX line printer. See `fpr` man page.
`fsplit`	Takes a Fortran multiple-routine source-code file and splits it into multiple files, one for each routine. See `fsplit` man page.
`ifnames`	Scans C source files and writes out a sorted list of all the identifiers that appear in `#if`, `#elif`, `#ifdef`, and `#ifndef` directives. See `ifnames` man page.
`indent`	Formats C source code. See `indent` man page.
`nmedit`	Changes global symbols in object code to static symbols. You can provide an input file that specifies which global symbols should remain global. The resulting object can still be used with the debugger. See `nmedit` man page.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Tool	Description
plutil	Can check the syntax of a property list or convert it from one format to another (XML or binary). See plutil man page.
printf	Formats and prints character strings and C constants. See printf man page.
ResMerger	Merges resources into resource files. When the Xcode application compiles Resource Manager resources, it sends them to a collector. After all Resource Manager resources have been compiled, the Xcode application calls ResMerger to put the resources in their final location. See ResMerger man page.
RezWack	Takes a compiled resource (.qtr) file and inserts it together with the data fork (.qtx or .exe file) into a Windows application (.exe) file. The resulting file is a Windows application that has the sort of resource fork that QuickTime understands. You can use the Rez tool to compile a resource source (.r) file. The RezWack tool is part of the QuickTime 3 Software Development Kit for Windows. See RezWack man page.
tops	Performs universal search and replace operations on text strings in source files. See tops man page.
unifdef	Removes #ifdef, #ifndef, #else, and #endif lines from code as specified in the input options. See unifdef man page.
UnRezWack	Reverses the effects of RezWack; that is, converts a single Windows executable file into separate data and resource files. See UnRezWack man page.

Debugging and Tuning Tools

Apple provides several tools for analyzing and monitoring the performance of your software. Performance should always be a key design goal of your programs. Using the provided tools, you can gather performance metrics and identify actual performance problems. You can then use this information to fix the problems and keep your software running efficiently.

General Tools

Table C-9 lists the command-line tools available for debugging. These tools are located in *<Xcode>*/usr/bin and /usr/bin.

Table C-9 General debugging tools

Tool	Description
defaults	Lets you read, write, and delete Mac OS X user defaults. A Mac OS X application uses the defaults system to record user preferences and other information that must be maintained when the application is not running. Not all these defaults are necessarily accessible through the application's preferences. See defaults man page.
gdb	The GNU debugger. You can use it through the Xcode application or can invoke it directly from the command line. See gdb man page.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Memory Analysis Tools

Table C-10 lists the applications and command-line tools for debugging and tuning memory problems. These tools are located in <*Xcode*>`/usr/bin` and `/usr/bin`.

Table C-10 Memory debugging and tuning tools

Tool	Description
`heap`	Lists all the objects currently allocated on the heap of the current process. It also describes any Objective-C objects, listed by class. See `heap` man page.
`leaks`	Examines a specified process for `malloc`-allocated buffers that are not referenced by the program. See `leaks` man page.
`malloc_history`	Inspects a given process and lists the `malloc` allocations performed by it. This tool relies on information provided by the standard `malloc` library when debugging options have been turned on. If you specify an address, `malloc_history` lists the allocations and deallocations that have manipulated a buffer at that address. For each allocation, a stack trace describing who called `malloc` or `free` is listed. See `malloc_history` man page.
`vmmap`	Displays the virtual memory regions allocated in a specified process, helping you understand how memory is being used and the purpose of memory (text segment, data segment, and so on) at a given address. See `vmmap` man page.
`vm_stat`	Displays Mach virtual memory statistics. See `vm_stat` man page.

Examining Code

Table C-11 lists the applications and command-line tools for examining generated code files. These tools are located in <*Xcode*>`/usr/bin` and `/usr/bin`.

Table C-11 Tools for examining code

Tool	Description
`c2ph`	Parses C code and outputs debugger information in the Stabs format, showing offsets of all the members of structures. For information on Stabs, see *STABS Debug Format*. See `c2ph` man page.
`cscope`	An interactive command-line tool that allows the user to browse through C source files for specified elements of code, such as functions, function calls, macros, variables, and preprocessor symbols. See `cscope` man page.
`ctags`	Makes a tags file for the `ex` line editor from specified C, Pascal, Fortran, YACC, Lex, or Lisp source files. A tags file lists the locations of symbols such as subroutines, typedefs, structs, enums, unions, and #defines. See `ctags` man page.
`error`	Analyzes error messages and can open a text editor to display the source of the error. The `error` tool is run with its input connected via a pipe to the output of the compiler or language processor generating the error messages. Note that the service provided by the `error` command is built into the Xcode application. See `error` man page.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Tool	Description
ibtool	Lets you print, update, and verify the contents of a nib file. You can use this tool to inject localized strings into a nib file or scan the contents of a nibfile using a script. (This tool replaces the nibtool program.) See ibtool man page.
nm	Displays the symbol tables of one or more object files, including the symbol type and value for each symbol. See nm man page.
otool	Displays specified parts of object files or libraries. See otool or otool64 man page.
pagestuff	Displays information about the specified logical pages of a file conforming to the Mach-O executable format. For each specified page of code, pagestuff displays symbols (function and static data structure names). See pagestuff man page.
pstruct	An alias to c2ph. See pstruct man page.
strings	Looks for ASCII strings in an object file or other binary file. See strings man page.

Performance Tools

Table C-12 lists the applications and command-line tools for analyzing and monitoring performance. For information about performance and the available performance tools, see *Performance Overview*. These tools are located in <*Xcode*>/usr/bin and /usr/bin.

Table C-12 Performance tools

Tool	Description
gprof	Produces an execution profile of a C, Pascal, or Fortran77 program. The tool lists the total execution times and call counts for each of the functions in the application, and sorts the functions according to the time they represent including the time of their call graph descendants. See gprof man page.
sample	Gathers data about the running behavior of a process. The sample tool stops the process at user-defined intervals, records the current function being executed by the process, and checks the stack to find how the current function was called. It then lets the application continue. At the end of a sampling session, sample produces a report showing which functions were executing during the session. See sample man page.
top	Displays an ongoing sample of system-use statistics. It can operate in various modes, but by default shows CPU and memory use for each process in the system. See top man page.

Instruction Trace Tools

Table C-13 lists the applications and command-line tools for working with hardware-level programs. These tools are located in /usr/bin.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Table C-13 Instruction trace tools

Tool	Description
acid	Analyzes TT6E (but not TT6) instruction traces and presents detailed analyses and histogram reports. See acid man page.
amber	Captures the instruction and data address stream generated by a process running in Mac OS X and saves it to disk in TT6, TT6E, or FULL format. Custom trace filters can be built using the amber_extfilt.a module in <*Xcode*>/Examples/CHUD/Amber/ExternalTraceFilter/. Differences between TT6 and TT6E format as well as the specifics of the FULL trace format are detailed in *Amber Trace Format Specification v1.1* (<*Xcode*>/ADC Reference Library/CHUD/AmberTraceFormats.pdf). See amber man page.
simg4	A cycle-accurate simulator of the Motorola 7400 processor that takes TT6 (not TT6E) traces as input. See simg4 man page.
simg5	A cycle-accurate simulator of the IBM 970 processor that takes TT6 (not TT6E) traces as input. See simg5 man page.

Documentation and Help Tools

Table C-14 lists applications and command-line tools for creating or working with documentation and online help. These tools are located in <*Xcode*>/usr/bin and /usr/bin.

Table C-14 Documentation and help tools

Tool	Description
compileHelp	Merges contextual help RTF snippets into one file. This tool is included to support legacy applications. New contextual help projects do not use this tool. See compileHelp man page.
gatherheaderdoc	Gathers HeaderDoc output, creating a single index page and cross-links between documents. See gatherheaderdoc man page.
headerdoc2HTML	Generates HTML documentation from structured commentary in C, C++, and Objective-C header files. The HeaderDoc tags and scripts are described at http://developer.apple.com/darwin/projects/headerdoc/. See headerdoc2html man page.
install-info	Inserts menu entries from an Info file into the top-level dir file in the GNU Texinfo documentation system. It's most often run as part of software installation or when constructing a dir file for all manuals on a system. See http://www.gnu.org/software/texinfo/manual/texinfo/ for more information on the GNU Texinfo system. See install-info man page.

Localization Tools

Table C-15 lists the applications and command-line tools for localizing your own applications. These tools are located in <*Xcode*>/usr/bin and /usr/bin.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Table C-15 Localization tools

Tool	Description
`DeRez`	Decompiles the resource fork of a resource file according to the type declarations in the type declaration files you specify. You can use this utility to find strings for localization purposes, for example. DeRez works with Resource Manager resource files, not with nib files.
`genstrings`	Takes the strings from C source code (`NSLocalizedString...`, `CFCopyLocalized-String...` functions) and generates string table files (`.strings` files). This tool can also work with `Bundle.localizedString...` methods in Java. See `genstrings` man page.
`Rez`	Compiles the resource fork of a file according to the textual description contained in the resource description files. You can use Rez to recompile the resource files you decompiled with DeRez after you have localized the strings.

Version Control Tools

Apple provides command-line tools to support several version-control systems. Unless otherwise noted, these tools are located in <*Xcode*>`/usr/bin` or `/usr/bin`.

Subversion

Table C-16 lists the command-line tools to use with the Subversion system.

Table C-16 Subversion tools

Tool	Description
`svn`	The Subversion command-line client tool. You use this tool for manipulating files in a Subversion archive. See `svn` man page.
`svnadmin`	Creates and manages Subversion repositories. See `svnadmin` man page.
`svndumpfilter`	Filters data dumped from the repository by a `svnadmin dump` command. See `svndumpfilter` man page.
`svnlook`	Examines repository revisions and transactions. See `svnlook` man page.
`svnserve`	Accesses a repository using the `svn` network protocol. See `svnserve` man page.
`svnversion`	Summarizes the revision mixture of a working copy. See `svnversion` man page.

RCS

Table C-17 lists the command-line tools to use with the RCS system.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Table C-17 RCS tools

Tool	Description
`ci`	Stores revisions in RCS files. If the RCS file doesn't exist, `ci` creates one. See `ci` man page.
`co`	Retrieves a revision from an RCS file and stores it in the corresponding working file. See `co` man page.
`rcs`	Creates new RCS files or changes attributes of existing ones. See `rcs` man page.
`rcs-checkin`	Checks a file into a new RCS file and uses the file's first line for the description.
`rcs2log`	Generates a change log from RCS files—which can possibly be located in a CVS repository—and sends the change log to standard output. See `rcs2log` man page.
`rcsclean`	Compares the working file to the latest revision (or a specified revision) in the corresponding RCS file and removes the working file if there is no difference. See `rcsclean` man page.
`rcsdiff`	Compares two revisions of an RCS file or the working file and one revision. See `rcsdiff` man page.
`rcsmerge`	Merges the changes in two revisions of an RCS file into the corresponding working file. See `rcsmerge` man page.

CVS

Table C-18 lists the command-line tools to use with the Concurrent Versions System (CVS) source control system.

Table C-18 CVS tools

Tool	Description
`agvtool`	Speeds up common versioning operations for Xcode projects that use the Apple-generic versioning system. It automatically embeds version information in the products produced by the Xcode application and performs certain CVS operations such as submitting the project with a new version number. For more information see the `agvtool` man page.
`cvs`	The latest tool for managing information in the CVS repository. (Note, this tool does not support CVS wrappers.) See the `cvs` man page for details. See also, `ocvs` below.
`cvs-wrap`	Wraps a directory into a GZIP format tar file. This single file can be handled more easily by CVS than the original directory.
`cvs-unwrap`	Extracts directories from a GZIP format tar file created by `cvs-wrap`.
`ocvs`	An older version of the `cvs` tool that still supports CVS wrappers. See the `ocvs` man page for details.

Comparing Files

Table C-19 lists the command-line tools for comparing files.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Table C-19 Comparison tools

Tool	Description
diff	Compares two files or the files in two directories. See diff man page.
diff3	Compares three files. See diff3 man page.
diffpp	Annotates the output of diff so that it can be printed with GNU enscript. This enables enscript to highlight the modified portions of the file. See diffpp man page.
diffstat	Reads one or more files output by diff and displays a histogram of the insertions, deletions, and modifications per file. See diffstat man page.
merge	Compares two files modified from the same original file and then combines all the changes into a single file. The merge tool warns you if both modified files have changes in the same lines. See merge man page.
opendiff	Opens FileMerge from the command line and begins comparing the specified files. See opendiff man page.
patch	Takes the output of diff and applies it to one or more copies of the original, unchanged file to create patched versions of the file. See patch man page.
sdiff	Compares two files and displays the differences so you can decide how to resolve them interactively. It then writes the results out to a file. A command-line version of FileMerge. See sdiff man page.

Packaging Tools

Table C-20 lists the applications and command-line tools used for packaging applications. These tools are located in <*Xcode*>/usr/bin and /usr/bin.

Table C-20 Packaging tools

Tool	Description
codesign	Creates a digital code signature for an application or software package. See codesign man page.
CpMac	Copies a file or a directory, including subdirectories, preserving metadata and forks. See CpMac man page.
GetFileInfo	Gets the file attributes of files in an HFS+ directory. See GetFileInfo man page.
install	Copies files to a target file or directory. Unlike the cp or mv commands, the install command lets you specify the new copy's owner, group ID, file flags, and mode. See install man page.
install_name_tool	Changes the dynamic shared library install names recorded in a Mach-O binary. See install_name_tool man page.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Tool	Description
lipo	Can create a multiple-architecture ("fat") executable file from one or more input files, list the architectures in a fat file, create a single-architecture file from a fat file, or make a new fat file with a subset of the architectures in the original fat file. See lipo man page.
MergePef	Merges two or more PEF files into a single file. PEF format is used for Mac OS 9 code. See MergePef man page.
mkbom	Creates a bill of materials for a directory.
MvMac	Moves files, preserving metadata and forks.
SetFile	Sets the attributes of files in an HFS+ directory. See SetFile man page.
SplitForks	Removes the resource fork in a file or all the resource forks in the files in a specified directory and saves them alongside the original files as hidden files (a hidden file has the same name as the original file, except that it has a "dot-underscore" prefix; for example ._MyPhoto.jpg.). See SplitForks man page.

Scripting Tools

The tools listed in the following sections are located in <*Xcode*>/usr/bin and /usr/bin.

Interpreters and Compilers

Table C-21 lists the command-line script interpreters and compilers.

Table C-21 Script interpreters and compilers

Tool	Description
awk	A pattern-directed scripting language for scanning and processing files. The scripting language is described on the awk man page.
osacompile	Compiles the specified files, or standard input, into a single script. Input files may be plain text or other compiled scripts. The osacompile command works with AppleScript and with any other OSA scripting language. See osacompile man page.
osascript	Executes a script file, which may be plain text or a compiled script. The osascript command works with AppleScript and with any other scripting language that conforms to the Open Scripting Architecture (OSA). See osascript man page.
perl	Executes scripts written in the Practical Extraction and Report Language (Perl). The man page for this command introduces the language and gives a list of other man pages that fully document it. See perl man page.
perlcc	Compiles Perl scripts. See perlcc man page.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Tool	Description
python	The interpreter for the Python language, an interactive, object-oriented language. Use the pydoc command to read documentation on Python modules. See python man page.
ruby	The interpreter for the Ruby language, an interpreted object-oriented scripting language. See ruby man page.
sed	Reads a set of files and processes them according to a list of commands. See sed man page.
tclsh	A shell-like application that interprets Tcl commands. It runs interactively if called without arguments. Tcl is a scripting language, like Perl, Python, or Ruby. However, Tcl is usually embedded and thus called from the Tcl library rather than by an interpreter such as tclsh. See tclsh man page.

Script Language Converters

Table C-22 lists the available command-line script language converters.

Table C-22 Script language converters

Tool	Description
a2p	Converts an awk script to a Perl script. See a2p man page.
s2p	Converts a sed script to a Perl script. See s2p man page.

Perl Tools

Table C-23 lists the available command-line Perl tools.

Table C-23 Perl tools

Tool	Description
dprofpp	Displays profile data generated for a Perl script by a Perl profiler. See dprofpp man page.
find2perl	Converts find command lines to equivalent Perl code. See find2perl man page.
h2ph	Converts C header files to Perl header file format. See h2ph man page.
h2xs	Builds a Perl extension from C header files. The extension includes functions that can be used to retrieve the value of any #define statement that was in the C header files. See h2xs man page.
perlbug	An interactive tool that helps you report bugs for the Perl language. See perlbug man page.
perldoc	Looks up and displays documentation for Perl library modules and other Perl scripts that include internal documentation. If a man page exists for the module, you can use man instead. See perldoc man page.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Tool	Description
pl2pm	Aids in the conversion of Perl 4 .pl library files to Perl 5 library modules. This tool is useful if you plan to update your library to use some of the features new in Perl 5. See pl2pm man page.
splain	Forces verbose warning diagnostics by the Perl compiler and interpreter. See splain man page.

Parsers and Lexical Analyzers

Table C-24 lists the available command-line parsers and lexical analyzers.

Table C-24 Parsers and lexical analyzers

Tool	Description
bison	Generates parsers from grammar specification files. A somewhat more flexible replacement for yacc. See bison man page.
flex	Generates programs that scan text files and perform pattern matching. When one of these programs matches the pattern, it executes the C routine you provide for that pattern. See flex man page.
lex	An alias for flex. See lex man page.
yacc	Generates parsers from grammar specification files. Used in conjunction with flex to created lexical analyzer programs. See yacc man page.

Documentation Tools

Table C-25 lists the available command-line scripting documentation tools.

Table C-25 Scripting documentation tools

Tool	Description
pod2html	Converts files from pod format to HTML format. The pod (Plain Old Documentation) format is defined in the perlpod man page. See pod2html man page.
pod2latex	Converts files from pod format to LaTeX format. LaTeX is a document preparation system built on the TeX text formatter. See pod2latex man page.
pod2man	Converts files from pod format to *roff code, which can be displayed using nroff via man, or printed using troff. See pod2man man page.
pod2text	Converts pod data to formatted ASCII text. See pod2text man page.
pod2usage	Similar to pod2text, but can output just the synopsis information or the synopsis plus any options/arguments sections instead of the entire man page. See pod2usage man page.
podchecker	Checks the syntax of documentation files that are in pod format and outputs errors to standard error. See podchecker man page.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Tool	Description
`podselect`	Prints selected sections of pod documentation to standard output. See `podselect` man page.

Java Tools

The tools listed in the following sections are located in `/usr/bin`.

General

Table C-26 lists the command-line tools used for building, debugging, and running Java programs.

Table C-26 Java tools

Tool	Description
`java`	Starts the Java runtime environment and launches a Java application. See `java` man page.
`javac`	The standard Java compiler from Sun Microsystems. See `javac` man page.
`jdb`	The Java debugger. It provides inspection and debugging of a local or remote Java virtual machine. See `jdb` man page.

Java Utilities

Table C-27 lists some of the applications and command-line tools for working with Java.

Table C-27 Java utilities

Tool	Description
`idlj`	Reads an Object Management Group (OMG) Interface Definition Language (IDL) file and translates it, or maps it, to a Java interface. The `idlj` compiler also creates stub, skeleton, helper, holder, and other files as necessary. These Java files are generated from the IDL file according to the mapping specified in the OMG document *OMG IDL to Java Language Mapping Specification, formal, 99-07-53*. The `idlj` compiler is documented at http://java.sun.com/j2se/1.3/docs/guide/rmi-iiop/toJavaPortableUG.html. IDL files are used to allow objects from different languages to interact with a common Object Request Broker (ORB), allowing remote invocation between languages. See `idlj` man page.
`javadoc`	Parses the declarations and documentation comments in a set of Java source files and produces HTML pages describing the public and protected classes, inner classes, interfaces, constructors, methods, and fields. See `javadoc` man page.
`javah`	Generates C header and source files from Java classes. The generated header and source files are used by C programs to reference instance variables of a Java object so that you can call Java code from inside your Mac OS X native application. See `javah` man page.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Tool	Description
native2ascii	Converts characters that are not in Latin-1 or Unicode encoding to ASCII for use with javac and other Java tools. It also can do the reverse conversion of Latin-1 or Unicode to native-encoded characters. See native2ascii man page.
rmic	A compiler that generates stub and skeleton class files for remote objects from the names of compiled Java classes that contain remote object implementations. A remote object is one that implements the interface java.rmi.Remote. See rmic man page.
rmiregistry	Creates and starts a remote object registry. A remote object registry is a naming service that makes it possible for clients on the host to look up remote objects and invoke remote methods. See rmiregistry man page.

Java Archive (JAR) Files

Table C-28 lists the available JAR file applications and command-line tools.

Table C-28 JAR file tools

Tool	Description
extcheck	Checks a specified JAR file for title and version conflicts with any extensions installed in the Java Developer Kit software. See extcheck man page.
jar	Combines and compresses multiple files into a single Java archive (JAR) file so they can be downloaded by a Java agent (such as a browser) in a single HTTP transaction. See jar man page.
jarsigner	Lets you sign JAR files and verify the signatures and integrity of signed JAR files. See jarsigner man page.

Kernel Extension Tools

Table C-29 lists the command-line tools that are useful for kernel extension development. These tools are located in /usr/sbin and /sbin.

Table C-29 Kernel extension tools

Tool	Description
kextload	Loads kernel extensions, validates them to make sure they can be loaded by other mechanisms, and generates symbol files for debugging them.
kextstat	Displays the status of any kernel extensions currently loaded in the kernel.
kextunload	Terminates and unregisters I/O Kit objects associated with a KEXT and unloads the code for the KEXT.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

I/O Kit Driver Tools

Table C-30 lists the applications and command-line tools for developing device drivers. These tools are located in <*Xcode*>`/usr/sbin`.

Table C-30 Driver tools

Tool	Description
`ioreg`	A command-line version of I/O Registry Explorer. The `ioreg` tool displays the tree in a Terminal window, allowing you to cut and paste sections of the tree.
`ioalloccount`	Displays a summary of memory allocated by I/O Kit allocators listed by type (instance, container, and `IOMalloc`). This tool is useful for tracking memory leaks.
`ioclasscount`	Shows the number of instances allocated for each specified class. This tool is also useful for tracking memory leaks.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Glossary

abstract type Defines, in information property lists, general characteristics of a family of documents. Each abstract type has corresponding concrete types. See also concrete type.

Accessibility The technology for ensuring that disabled users can use Mac OS X. Accessibility provides support for disabled users in the form of screen readers, speech recognition, text-to-speech converters, and mouse and keyboard alternatives.

ACLs Access Control Lists. A technology used to give more fine-grained access to file-system objects. Compare with permissions.

actions Building blocks used to build workflows in Automator.

active window The frontmost modal or document window. Only the contents of the active window are affected by user actions. The active window has distinctive details that aren't visible for inactive windows.

Address Book A technology for managing names, addresses, phone numbers, and other contact-related information. Mac OS X provides the Address Book application for managing contact data. It also provides the Address Book framework so that applications can programmatically manage the data.

address space Describes the range of memory (both physical and virtual) that a process uses while running. In Mac OS X, processes do not share address space.

alias A lightweight reference to files and folders in Mac OS Standard (HFS) and Mac OS Extended (HFS+) file systems. An alias allows multiple references to files and folders without requiring multiple copies of these items. Aliases are not as fragile as symbolic links because they identify the volume and location on disk of a referenced file or folder; the file or folder can be moved around without breaking the alias. See also symbolic link.

anti-aliasing A technique that smoothes the roughness in images or sound caused by aliasing. During frequency sampling, aliasing generates a false (alias) frequency along with the correct one. With images this produces a stair-step effect. Anti-aliasing corrects this by adjusting pixel positions or setting pixel intensities so that there is a more gradual transition between pixels.

Apple event A high-level operating-system event that conforms to the Apple Event Interprocess Messaging Protocol (AEIMP). An Apple event typically consists of a message from an application to itself or to another application.

AppleScript An Apple-defined scripting language. AppleScript uses a natural language syntax to send Apple events to applications, commanding them to perform specific actions.

AppleTalk A suite of network protocols that is standard on Macintosh computers and can be integrated with other network systems, such as the Internet.

Application Kit A Cocoa framework that implements an application's user interface. The Application Kit provides a basic program structure for applications that draw on the screen and respond to events.

application packaging Putting code and resources in the prescribed directory locations inside application bundles. "Application package" is sometimes used synonymously with "application bundle."

Aqua A set of guidelines that define the appearance and behavior of Mac OS X applications. The Aqua guidelines bring a unique look to applications,

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

integrating color, depth, clarity, translucence, and motion to present a vibrant appearance. If you use Carbon, Cocoa, or X11 to create your application's interface, you get the Aqua appearance automatically

ASCII American Standard Code for Information Interchange. A 7-bit character set (commonly represented using 8 bits) that defines 128 unique character codes. See also Unicode.

bit depth The number of bits used to describe something, such as the color of a pixel. Each additional bit in a binary number doubles the number of possibilities.

bitmap A data structure that represents the positions and states of a corresponding set of pixels.

Bonjour Apple's technology for zero-configuration networking. Bonjour enables dynamic discovery of services over a network.

BSD Berkeley Software Distribution. Formerly known as the Berkeley version of UNIX, BSD is now simply called the BSD operating system. BSD provides low-level features such as networking, thread management, and process communication. It also includes a command-shell environment for managing system resources. The BSD portion of Mac OS X is based on version 5 of the FreeBSD distribution.

buffered window A window with a memory buffer into which all drawing is rendered. All graphics are first drawn in the buffer, and then the buffer is flushed to the screen.

bundle A directory in the file system that stores executable code and the software resources related to that code. Applications, plug-ins, and frameworks are types of bundles. Except for frameworks, bundles are file packages, presented by the Finder as a single file.

bytecode Computer object code that is processed by a virtual machine. The virtual machine converts generalized machine instructions into specific machine instructions (instructions that a computer's processor can understand). Bytecode is the result of compiling source language statements written in any language that supports this approach. The best-known language today that uses the bytecode and virtual machine approach is Java. In Java, bytecode is contained in a binary file with a `.class` suffix. (Strictly speaking, "bytecode" means that the individual instructions are one byte long, as opposed to PowerPC code, for example, which is four bytes long.) See also virtual machine (VM).

Carbon An application environment in Mac OS X that features a set of procedural programming interfaces derived from earlier versions of the Mac OS. The Carbon API has been modified to work properly with Mac OS X, especially with the foundation of the operating system, the kernel environment. Carbon applications can run in Mac OS X and Mac OS 9.

CFM Code Fragment Manager, the library manager and code loader for processes based on PEF (Preferred Executable Format) object files (in Carbon).

class In object-oriented languages such as Java and Objective-C, a prototype for a particular kind of object. A class definition declares instance variables and defines methods for all members of the class. Objects that belong to the same class have the same types of instance variables and have access to the same methods (included the instance variables and methods inherited from superclasses).

Classic An application environment in Mac OS X that lets you run non-Carbon legacy Mac OS software. It supports programs built for both PowerPC and 68000-family chip architectures and is fully integrated with the Finder and the other application environments.

Clipboard A per-user server (also known as the pasteboard) that enables the transfer of data between applications, including the Finder. This server is shared by all running applications and contains data that the user has cut or copied, as well as other data that one application wants to transfer to another, such as in dragging operations. Data in the Clipboard is associated with a name that indicates how it is to be used. You implement data-transfer operations with the Clipboard using Core Foundation Pasteboard Services or the Cocoa NSPasteboard class. See also pasteboard.

Cocoa An advanced object-oriented development platform in Mac OS X. Cocoa is a set of frameworks used for the rapid development of full-featured applications in the Objective-C language. It is based on the integration of OpenStep, Apple technologies, and Java.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

code fragment In the CFM-based architecture, a code fragment is the basic unit for executable code and its static data. All fragments share fundamental properties such as the basic structure and the method of addressing code and data. A fragment can easily access code or data contained in another fragment. In addition, fragments that export items can be shared among multiple clients. A code fragment is structured according to the Preferred Executable Format (PEF).

ColorSync An industry-standard architecture for reliably reproducing color images on various devices (such as scanners, video displays, and printers) and operating systems.

compositing A method of overlaying separately rendered images into a final image. It encompasses simple copying as well as more sophisticated operations that take advantage of transparency.

concrete type Defines, in information property lists, specific characteristics of a type of document, such as extensions and HFS+ type and creator codes. Each concrete type has corresponding abstract types. See also abstract type.

cooperative multitasking A multitasking environment in which a running program can receive processing time only if other programs allow it; each application must give up control of the processor "cooperatively" in order to allow others to run. Mac OS 9 is a cooperative multitasking environment. See also preemptive multitasking.

CUPS The Common UNIX Printing System; an open source architecture commonly used by the UNIX community to implement printing.

daemon A process that handles periodic service requests or forwards a request to another process for handling. Daemons run continuously, usually in the background, waking only to handle their designated requests. For example, the `httpd` daemon responds to HTTP requests for web information.

Darwin Another name for the Mac OS X core operating system. The Darwin kernel is equivalent to the Mac OS X kernel plus the BSD libraries and commands essential to the BSD Commands environment. Darwin is an open source technology.

Dashboard A user technology for managing HTML-based programs called widgets (see permissions). Activating the Dashboard via the F12 key displays a layer above the Mac OS X desktop that contains the user's current set of widgets.

Dashcode A graphical application used to build and debug Dashboard widgets.

demand paging An operating system facility that causes pages of data to be read from disk into physical memory only as they are needed.

device driver A component of an operating system that deals with getting data to and from a device, as well as the control of that device.

domain An area of the file system reserved for software, documents, and resources and limiting the accessibility of those items. A domain is segregated from other domains. There are four domains: user, local, network, and system.

DVD An optical storage medium that provides greater capacity and bandwidth than CD-ROM; DVDs are frequently used for multimedia as well as data storage.

dyld See dynamic link editor.

dynamic link editor The library manager for code in the Mach-O executable format. The dynamic link editor is a dynamic library that "lives" in all Mach-O programs on the system. See also CFM; Mach-O.

dynamic linking The binding of modules, as a program executes, by the dynamic link editor. Usually the dynamic link editor binds modules into a program lazily (that is, as they are used). Thus modules not actually used during execution are never bound into the program.

dynamic shared library A library whose code can be shared by multiple, concurrently running programs. Programs share exactly one physical copy of the library code and do not require their own copies of that code. With dynamic shared libraries, a program not only attempts to resolve all undefined symbols at runtime, but attempts to do so only when those symbols are referenced during program execution.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

encryption The conversion of data into a form, called ciphertext, that cannot be easily understood by unauthorized people. The complementary process, decryption, converts encrypted data back into its original form.

Ethernet A high-speed local area network technology.

exception An interruption to the normal flow of program control that occurs when an error or other special condition is detected during execution. An exception transfers control from the code generating the exception to another piece of code, generally a routine called an exception handler.

fault In the virtual-memory system, faults are the mechanism for initiating page-in activity. They are interrupts that occur when code tries to access data at a virtual address that is not mapped to physical memory. Soft faults happen when the referenced page is resident in physical memory but is unmapped. Hard (or page) faults occur when the page has been swapped out to backing store. See also page; virtual memory.

file package A directory that the Finder presents to users as if it were a file. In other words, the Finder hides the contents of the directory from users. This opacity discourages users from inadvertently (or intentionally) altering the contents of the directory.

file system A part of the kernel environment that manages the reading and writing of data on mounted storage devices of a certain volume format. A file system can also refer to the logical organization of files used for storing and retrieving them. File systems specify conventions for naming files, storing data in files, and specifying locations of files. See also volume format.

filters The simplest unit used to modify image data from Core Image. One or more filters may be packaged into an image units and loaded into a program using the Core image framework. Filters can contain executable or nonexecutable code.

firewall Software (or a computer running such software) that prevents unauthorized access to a network by users outside the network. (A physical firewall prevents the spread of fire between two physical locations; the software analog prevents the unauthorized spread of data.)

fork (1) A stream of data that can be opened and accessed individually under a common filename. The Mac OS Standard and Extended file systems store a separate data fork and resource fork as part of every file; data in each fork can be accessed and manipulated independently of the other. (2) In BSD, `fork` is a system call that creates a new process.

framebuffer A highly accessible part of video RAM (random-access memory) that continuously updates and refreshes the data sent to the devices that display images onscreen.

framework A type of bundle that packages a dynamic shared library with the resources that the library requires, including header files and reference documentation.

HFS Hierarchical File System. The Mac OS Standard file-system format, used to represent a collection of files as a hierarchy of directories (folders), each of which may contain either files or other folders. HFS is a two-fork volume format.

HFS+ Hierarchical File System Plus. The Mac OS Extended file-system format. This format was introduced as part of Mac OS 8.1, adding support for filenames longer than 31 characters, Unicode representation of file and directory names, and efficient operation on very large disks. HFS+ is a multiple-fork volume format.

HIToolbox Human Interface Toolbox. A collection of procedural APIs that apply an object-oriented model to windows, controls, and menus for Carbon applications. The HI Toolbox supplements older Macintosh Toolbox managers such as the Control Manager, Dialog Manager, Menu Manager, and Window Manager from Mac OS 9.

host The computer that is running (is host to) a particular program; used to refer to a computer on a network.

IDE An acronym meaning "integrated development environment". An IDE is a program that typically combines text editing, compiling, and debugging features in one package in order to assist developers with the creation of software.

image units A plug-in bundle for use with the Core Image framework. Image units contain one or more filters for manipulating image data.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

information property list A property list that contains essential configuration information for bundles. A file named `Info.plist` (or a platform-specific variant of that filename) contains the information property list and is packaged inside the bundle.

inheritance In object-oriented programming, the ability of a superclass to pass its characteristics (methods and instance variables) on to its subclasses.

instance In object-oriented languages such as Java and Objective-C, an object that belongs to (is a member of) a particular class. Instances are created at runtime according to the specification in the class definition.

Interface Builder A tool for creating user interfaces. You use this tool to build and configure your user interface using a set of standard components and save that data to a resource file that can be loaded into your program at runtime. For more information, see "Interface Builder" (page 133).

internationalization The design or modification of a software product, including its online help and documentation, to facilitate localization. Internationalization of software typically involves writing or modifying code to make use of locale-aware operating-system services for appropriate localized text input, display, formatting, and manipulation. See also localization.

interprocess communication (IPC) A set of programming interfaces that enables a process to communicate data or information to another process. Mechanisms for IPC exist in the different layers of the system, from Mach messaging in the kernel to distributed notifications and Apple events in the application environments. Each IPC mechanism has its own advantages and limitations, so it is not unusual for a program to use multiple IPC mechanisms. Other IPC mechanisms include pipes, named pipes, signals, message queueing, semaphores, shared memory, sockets, the Clipboard, and application services.

I/O Kit A collection of frameworks, libraries, tools, and other resources for creating device drivers in Mac OS X. The I/O Kit framework uses a restricted form of C++ to provide default behavior and an object-oriented programming model for creating custom drivers.

iSync A tool for synchronizing address book information.

Java A development environment for creating applications. Java was created by Sun Microsystems.

Java Native Interface (JNI) A technology for bridging C-based code with Java.

Java Virtual Machine (JVM) The runtime environment for executing Java code. This environment includes a just-in-time bytecode compiler and utility code.

kernel The complete Mac OS X core operating-system environment, which includes Mach, BSD, the I/O Kit, file systems, and networking components. Also called the kernel environment.

key An arbitrary value (usually a string) used to locate a piece of data in a data structure such as a dictionary.

localization The adaptation of a software product, including its online help and documentation, for use in one or more regions of the world, in addition to the region for which the original product was created. Localization of software can include translation of user interface text, resizing of text-related graphical elements, and replacement or modification of user interface images and sound. See also internationalization.

lock A data structure used to synchronize access to a shared resource. The most common use for a lock is in multithreaded programs where multiple threads need access to global data. Only one thread can hold the lock at a time; this thread is the only one that can modify the data during this period.

manager In Carbon, a library or set of related libraries that define a programming interface.

Mach The lowest level of the Mac OS X kernel environment. Mach provides such basic services and abstractions as threads, tasks, ports, interprocess communication (IPC), scheduling, physical and virtual address space management, virtual memory, and timers.

Mach-O Executable format of Mach object files. See also PEF.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

main thread By default, a process has one thread, the main thread. If a process has multiple threads, the main thread is the first thread in the process. A user process can use the POSIX threading API (pthread) to create other user threads.

major version A framework version specifier designating a framework that is incompatible with programs linked with a previous version of the framework's dynamic shared library.

makefile A specification file used by a build tool to create an executable version of an application. A makefile details the files, dependencies, and rules by which the application is built.

memory-mapped file A file whose contents are mapped into memory. The virtual-memory system transfers portions of these contents from the file to physical memory in response to page faults. Thus, the disk file serves as backing store for the code or data not immediately needed in physical memory.

memory protection A system of memory management in which programs are prevented from being able to modify or corrupt the memory partition of another program. Mac OS 9 does not have memory protection; Mac OS X does.

method In object-oriented programming, a procedure that can be executed by an object.

minor version A framework version specifier designating a framework that is compatible with programs linked with later builds of the framework within the same major version.

multicast A process in which a single network packet may be addressed to multiple recipients. Multicast is used, for example, in streaming video, in which many megabytes of data are sent over the network.

multihoming The ability to have multiple network addresses in one computer. For example, multihoming might be used to create a system in which one address is used to talk to hosts outside a firewall and the other to talk to hosts inside; the operating system provides facilities for passing information between the two.

multitasking The concurrent execution of multiple programs. Mac OS X uses preemptive multitasking, whereas Mac OS 9 uses cooperative multitasking.

network A group of hosts that can directly communicate with each other.

nib file A file containing resource data generated by the Interface Builder application.

nonretained window A window without an offscreen buffer for screen pixel values.

notification Generally, a programmatic mechanism for alerting interested recipients (or "observers") that some event has occurred during program execution. The observers can be users, other processes, or even the same process that originates the notification. In Mac OS X, the term "notification" is used to identify specific mechanisms that are variations of the basic meaning. In the kernel environment, "notification" is sometimes used to identify a message sent via IPC from kernel space to user space; an example of this is an IPC notification sent from a device driver to the window server's event queue. Distributed notifications provide a way for a process to broadcast an alert (along with additional data) to any other process that makes itself an observer of that notification. Finally, the Notification Manager (a Carbon manager) lets background programs notify users—through blinking icons in the menu bar, by sounds, or by dialogs—that their intercession is required.

NFS Network File System. An NFS file server allows users on the network to share files on other hosts as if they were on their own local disks.

object A programming unit that groups together a data structure (instance variables) and the operations (methods) that can use or affect that data. Objects are the principal building blocks of object-oriented programs.

object file A file containing executable code and data. Object files in the Mach-O executable format take the suffix `.o` and are the product of compilation using the GNU compiler (`gcc`). Multiple object files are typically linked together along with required frameworks to create a program. See also code fragment; dynamic linking.

object wrapper Code that defines an object-based interface for a set of procedural interfaces. Some Cocoa objects wrap Carbon interfaces to provide parallel functionality between Cocoa and Carbon applications.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Objective-C An object-oriented programming language based on standard C and a runtime system that implements the dynamic functions of the language. Objective-C's few extensions to the C language are mostly based on Smalltalk, one of the first object-oriented programming languages. Objective-C is available in the Cocoa application environment.

opaque type In Core Foundation and Carbon, an aggregate data type plus a suite of functions that operate on instances of that type. The individual fields of an initialized opaque type are hidden from clients, but the type's functions offer access to most values of these fields. An opaque type is roughly equivalent to a class in object-oriented programming.

OpenGL The Open Graphics Language; an industry-wide standard for developing portable 2D and 3D graphics applications. OpenGL consists of an API and libraries that developers use to render content in their applications.

open source A definition of software that includes freely available access to source code, redistribution, modification, and derived works. The full definition is available at `www.opensource.org`.

Open Transport Open Transport is a legacy communications architecture for implementing network protocols and other communication features on computers running the Mac OS. Open Transport provides a set of programming interfaces that supports, among other things, both the AppleTalk and TCP/IP protocols.

package In Java, a way of storing, organizing, and categorizing related Java class files; typical package names are `java.util` and `com.apple.cocoa.foundation`. See also application packaging.

PackageMaker A tool that builds an installable software package from the files you provide. For more information, see "PackageMaker" (page 142).

page The smallest unit, measured in bytes, of information that the virtual memory system can transfer between physical memory and backing store. As a verb, page refers to transferring pages between physical memory and backing store.

pasteboard Another name for the Clipboard.

PEF Preferred Executable Format. An executable format understood by the Code Fragment Manager. See also Mach-O.

permissions In BSD, a set of attributes governing who can read, write, and execute resources in the file system. The output of the `ls -l` command represents permissions as a nine-position code segmented into three binary three-character subcodes; the first subcode gives the permissions for the owner of the file, the second for the group that the file belongs to, and the last for everyone else. For example, `-rwxr-xr--` means that the owner of the file has read, write, execute permissions (rwx); the group has read and execute permissions (r-x); everyone else has only read permissions. (The leftmost position indicates whether this is a regular file (-), a directory (d), a symbolic link (l), or a special pseudo-file device.) The execute bit has a different semantic for directories, meaning they can be searched.

physical address An address to which a hardware device, such as a memory chip, can directly respond. Programs, including the Mach kernel, use virtual addresses that are translated to physical addresses by mapping hardware controlled by the Mach kernel.

physical memory Electronic circuitry contained in random-access memory (RAM) chips, used to temporarily hold information at execution time. Addresses in a process's virtual memory are mapped to addresses in physical memory. See also virtual memory.

pixel The basic logical unit of programmable color on a computer display or in a computer image. The physical size of a pixel depends on the resolution of the display screen.

plug-in An external module of code and data separate from a host (such as an application, operating system, or other plug-in) that, by conforming to an interface defined by the host, can add features to the host without needing access to the source code of the host. Plug-ins are types of loadable bundles. They are implemented with Core Foundation Plug-in Services.

port (1) In Mach, a secure unidirectional channel for communication between tasks running on a single system. (2) In IP transport protocols, an integer identifier used to select a receiver for an incoming packet or to specify the sender of an outgoing packet.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

POSIX The Portable Operating System Interface. An operating-system interface standardization effort supported by ISO/IEC, IEEE, and The Open Group.

PostScript A language that describes the appearance (text and graphics) of a printed page. PostScript is an industry standard for printing and imaging. Many printers contain or can be loaded with PostScript software. PostScript handles industry-standard, scalable typefaces in the Type 1 and TrueType formats. PostScript is an output format of Quartz.

preemption The act of interrupting a currently running task in order to give time to another task.

preemptive multitasking A type of multitasking in which the operating system can interrupt a currently running task in order to run another task, as needed. See also cooperative multitasking.

process A BSD abstraction for a running program. A process's resources include a virtual address space, threads, and file descriptors. In Mac OS X, a process is based on one Mach task and one or more Mach threads.

property list A structured, textual representation of data that uses the Extensible Markup Language (XML) as the structuring medium. Elements of a property list represent data of certain types, such as arrays, dictionaries, and strings.

pthreads The POSIX Threads package (BSD).

Quartz The native 2D rendering API for Mac OS X. Quartz contains programmatic interfaces that provide high-quality graphics, compositing, translucency, and other effects for rendered content. Quartz is included as part of the Application Services umbrella framework.

Quartz Extreme A technology integrated into the lower layers of Quartz that enables many graphics operations to be offloaded to hardware. This offloading of work to the graphics processor unit (GPU) provides tremendous acceleration for graphics-intensive applications. This technology is enabled automatically by Quartz and OpenGL on supported hardware.

QuickTime Apple's multimedia authoring and rendering technology. QuickTime lets you import and export media files, create new audio and video content, modify existing content, and play back content.

RAM Random-access memory. Memory that a microprocessor can either read or write to.

raster graphics Digital images created or captured (for example, by scanning in a photo) as a set of samples of a given space. A raster is a grid of x-axis (horizontal) and y-axis (vertical) coordinates on a display space. (Three-dimensional images also have a z coordinate.) A raster image identifies the monochrome or color value with which to illuminate each of these coordinates. The raster image is sometimes referred to as a bitmap because it contains information that is directly mapped to the display grid. A raster image is usually difficult to modify without loss of information. Examples of raster-image file types are BMP, TIFF, GIF, and JPEG files. See also vector graphics.

real time In reference to operating systems, a guarantee of a certain capability within a specified time constraint, thus permitting predictable, time-critical behavior. If the user defines or initiates an event and the event occurs instantaneously, the computer is said to be operating in real time. Real-time support is especially important for multimedia applications.

reentrant The ability of code to process multiple interleaved requests for service nearly simultaneously. For example, a reentrant function can begin responding to one call, be interrupted by other calls, and complete them all with the same results as if the function had received and executed each call serially.

resolution The number of pixels (individual points of color) contained on a display monitor, expressed in terms of the number of pixels on the horizontal axis and the number on the vertical axis. The sharpness of the image on a display depends on the resolution and the size of the monitor. The same resolution will be sharper on a smaller monitor and gradually lose sharpness on larger monitors because the same number of pixels are being spread out over a larger area.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

resource Anything used by executable code, especially by applications. Resources include images, sounds, icons, localized strings, archived user interface objects, and various other things. Mac OS X supports both Resource Manager–style resources and "per-file" resources. Localized and nonlocalized resources are put in specific places within bundles.

retained window A window with an offscreen buffer for screen pixel values. Images are rendered into the buffer for any portions of the window that aren't visible onscreen.

role An identifier of an application's relation to a document type. There are five roles: Editor (reads and modifies), Viewer (can only read), Print (can only print), Shell (provides runtime services), and None (declares information about type). You specify document roles in an application's information property list.

ROM Read-only memory, that is, memory that cannot be written to.

run loop The fundamental mechanism for event monitoring in Mac OS X. A run loop registers input sources such as sockets, Mach ports, and pipes for a thread; it also enables the delivery of events through these sources. In addition to sources, run loops can also register timers and observers. There is exactly one run loop per thread.

runtime The period of time during which a program is being executed, as opposed to compile time or load time. Can also refer to the runtime environment, which designates the set of conventions that arbitrate how software is generated into executable code, how code is mapped into memory, and how functions call one another.

Safari Apple's web browser. Safari is the default web browser that ships with Mac OS X.

scheduling The determination of when each process or task runs, including assignment of start times.

SCM Repository Source Code Management Repositories. A code database used to enable the collaborative development of large projects by multiple engineers. SCM repositories are managed by specific tools (such as CVS and Subversion), which manage the repository and handle check-ins and check-outs of code resources by engineers.

SCSI Small Computer Systems Interface. A standard connector and communications protocol used for connecting devices such as disk drives to computers.

script A series of statements, written in a scripting language such as AppleScript or Perl, that instruct an application or the operating system to perform various operations. Interpreter programs translate scripts.

semaphore A programming technique for coordinating activities in which multiple processes compete for the same kernel resources. Semaphores are commonly used to share a common memory space and to share access to files. Semaphores are one of the techniques for interprocess communication in BSD.

server A process that provides services to other processes (clients) in the same or other computers.

Shark A tool for analyzing a running (or static) application that returns metrics to help you identify potential performance bottlenecks. For more information, see "Performance Tools" (page 148).

sheet A dialog associated with a specific window. Sheets appear to slide out from underneath the window title and float above the window.

shell An interactive programming language interpreter that runs in a Terminal window. Mac OS X includes several different shells, each with a specialized syntax for executing commands and writing structured programs, called shell scripts.

SMP Symmetric multiprocessing. A feature of an operating system in which two or more processors are managed by one kernel, sharing the same memory and having equal access to I/O devices, and in which any task, including kernel tasks, can run on any processor.

socket (1) In BSD-derived systems, a socket refers to different entities in user and kernel operations. For a user process, a socket is a file descriptor that has been allocated using `socket(2)`. For the kernel, a socket is the data structure that is allocated when the kernel's implementation of the `socket(2)` call is made. (2) In AppleTalk protocols, a socket serves the same purpose as a "port" in IP transport protocols.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

spool To send files to a device or program (called a spooler or daemon) that puts them in a queue for later processing. The print spooler controls output of jobs to a printer. Other devices, such as plotters and input devices, can also have spoolers.

subframework A public framework that packages a specific Apple technology and is part of an umbrella framework. Through various mechanisms, Apple prevents or discourages developers from including or directly linking with subframeworks. See also umbrella framework.

symbolic link A lightweight reference to files and folders in UFS file systems. A symbolic link allows multiple references to files and folders without requiring multiple copies of these items. Symbolic links are fragile because if what they refer to moves somewhere else in the file system, the link breaks. However, they are useful in cases where the location of the referenced file or folder will not change. See also alias.

system framework A framework developed by Apple and installed in the file-system location for system software.

task A Mach abstraction, consisting of a virtual address space and a port name space. A task itself performs no computation; rather, it is the context in which threads run. See also thread.

TCP/IP Transmission Control Protocol/Internet Protocol. An industry-standard protocol used to deliver messages between computers over the network. TCP/IP support is included in Mac OS X.

thread In Mach, the unit of CPU utilization. A thread consists of a program counter, a set of registers, and a stack pointer. See also task.

thread-safe code Code that can be used safely by several threads simultaneously.

timer A kernel resource that triggers an event at a specified interval. The event can occur only once or can be recurring. Timers are one of the input sources for run loops. Timers are also implemented at higher levels of the system, such as CFTimer in Core Foundation and NSTimer in Cocoa.

transformation An alteration to a coordinate system that defines a new coordinate system. Standard transformations include rotation, scaling, and translation. A transformation is represented by a matrix.

UDF Universal Disk Format. The file-system format used in DVD disks.

UFS UNIX file system. An industry-standard file-system format used in UNIX-like operating systems such as BSD. UFS in Mac OS X is a derivative of 4.4BSD UFS. Its disk layout is not compatible with other BSD UFS implementations.

umbrella framework A system framework that includes and links with constituent subframeworks and other public frameworks. An umbrella framework "contains" the system software defining an application environment or a layer of system software. See also subframework.

Unicode A 16-bit character set that assigns unique character codes to characters in a wide range of languages. In contrast to ASCII, which defines 128 distinct characters typically represented in 8 bits, Unicode comprises 65536 distinct characters that represent the unique characters used in many languages.

vector graphics The creation of digital images through a sequence of commands or mathematical statements that place lines and shapes in a two-dimensional or three-dimensional space. One advantage of vector graphics over bitmap graphics (or raster graphics) is that any element of the picture can be changed at any time because each element is stored as an independent object. Another advantage of vector graphics is that the resulting image file is typically smaller than a bitmap file containing the same image. Examples of vector-image file types are PDF, encapsulated PostScript (EPS), and SVG. See also raster graphics.

versioning With frameworks, schemes to implement backward and forward compatibility of frameworks. Versioning information is written into a framework's dynamic shared library and is also reflected in the internal structure of a framework. See also major version; minor version.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

VFS Virtual File System. A set of standard internal file-system interfaces and utilities that facilitate support for additional file systems. VFS provides an infrastructure for file systems built into the kernel.

virtual address A memory address that is usable by software. Each task has its own range of virtual addresses, which begins at address zero. The Mach operating system makes the CPU hardware map these addresses onto physical memory only when necessary, using disk memory at other times. See also physical address.

virtual machine (VM) A simulated computer in that it runs on a host computer but behaves as if it were a separate computer. The Java virtual machine works as a self-contained operating environment to run Java applications and applets.

virtual memory The use of a disk partition or a file on disk to provide the facilities usually provided by RAM. The virtual-memory manager in Mac OS X provides either a 32-bit or 64-bit protected address space for each task (depending on the options used to build the task) and facilitates efficient sharing of that address space.

VoiceOver A spoken user interface technology for visually impaired users.

volume A storage device or a portion of a storage device that is formatted to contain folders and files of a particular file system. A hard disk, for example, may be divided into several volumes (also known as partitions).

volume format The structure of file and folder (directory) information on a hard disk, a partition of a hard disk, a CD-ROM, or some other volume mounted on a computer system. Volume formats can specify such things as multiple forks (HFS and HFS+), symbolic and hard links (UFS), case sensitivity of filenames, and maximum length of filenames. See also file system.

widget An HTML-based program that runs in the Dashboard layer of the system.

window server A systemwide process that is responsible for rudimentary screen displays, window compositing and management, event routing, and cursor management. It coordinates low-level windowing behavior and enforces a fundamental uniformity in what appears on the screen.

Xcode An integrated development environment (IDE) for creating Mac OS X software. Xcode incorporates compiler, debugger, linker, and text editing tools into a single package to streamline the development process. For more information, see "Xcode" (page 127).

Instruments An integrated performance analysis and debugging tool. Instruments lets you gather a configurable set of metrics while your application is running, providing you with visualization tools to analyze the data and see performance problems and potential coding errors within your software. For more information, see "Instruments" (page 135).

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *Mac OS X Technology Overview*.

Date	Notes
2008-10-15	Removed outdated reference to jikes compiler. Marked the AppleShareClient framework as deprecated, which it was in Mac OS X v10.5.
2007-10-31	Updated for Mac OS X v10.5. The document was also reorganized.
2006-06-28	Associated in-use prefix information with the system frameworks. Clarified directories containing developer tools.
2005-10-04	Added references to "Universal Binary Programming Guidelines."
2005-08-11	Fixed minor typos. Updated environment variable inheritance information.
2005-07-07	Incorporated developer feedback.
	Added AppleScript to the list of application environments.
2005-06-04	Corrected the man page name for SQLite.
2005-04-29	Fixed broken links and incorporated user feedback.
	Incorporated porting and technology guidelines from "Apple Software Design Guidelines." Added information about new system technologies. Changed "Rendezvous" to "Bonjour."
	Added new software types to list of development opportunities.
	Added a command-line primer.
	Added a summary of the available development tools.
	Updated the list of system frameworks.
2004-05-27	First version of *Mac OS X Technology Overview*. Some of the information in this document previously appeared in *System Overview*.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Index

Symbols

> operator 111
< operator 111
| operator 111

Numerals

3D graphics 119
64-bit support 30
802.1x protocol 25

A

a2p tool 154
Abstract Windowing Toolkit package 39
Accelerate framework 96
Accelerate.framework 53, 120
access control lists 23
Accessibility Inspector 140
Accessibility Verifier 140
accessibility
  support for 75–76
  technologies 97
acid tool 149
ACLs. *See* access control lists
adaptability
  explained 100
  technologies for implementing 100
ADC. *See* Apple Developer Connection
Address Book 59
Address Book action plug-ins 82
AddressBook.framework 115
AE.framework 123
agent applications 86
AGL.framework 115
AGP 22, 92
agvtool tool 151
AirPort 26
AirPort Extreme 26
amber tool 149
anti-aliasing 42
Apache HTTP server 27, 88
AppKit.framework 115
AppKitScripting.framework 115
Apple Developer Connection (ADC) 14
Apple events 36, 123
Apple Guide 105
Apple Information Access Toolkit 65
Apple Type Services 47, 121
Apple Type Services for Unicode Imaging. *See* ATSUI
AppleScript application environment 57
AppleScript Studio 57, 136
AppleScript
  overview 76
  script language 91
  scripting additions 91
  web services 88
  when to use 97, 101
AppleScriptKit.framework 115
AppleShareClient.framework 115
AppleShareClientCore.framework 115
Applet Launcher 138
AppleTalk 26
AppleTalk Filing Protocol (AFP) 24
AppleTalk Manager 105
AppleTalk.framework 116
Application Kit 56
application plug-ins 82
application services 87
applications
  and interapplication communication 36
  bundling 73
  opening 64
ApplicationServices.framework 56, 121
Aqua 71, 97, 98
architecture
  hardware 29
as tool 144
assistive devices 76

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

ATS. *See* Apple Type Services
ATS.framework 121
ATSUI 48
attractive appearance
  explained 98
  technologies for implementing 98
AU Lab 137
audio units 137
audio
  delivery 49
  file formats 50
AudioToolbox.framework 116
AudioUnit.framework 116
authentication 66, 118
Authorization Services 66, 99
Automator 59, 77, 83, 116
Automator.framework 121
availability of APIs 96
awk tool 153

B

bash shell 91
Berkeley Software Distribution. *See* BSD
BigTop 139
bison tool 155
Bluetooth 118
Bluetooth Explorer 140
Bonjour 27, 59–60, 97
Bootstrap Protocol (BOOTP) 25
BSD
  application environment 58
  command line interface 109
  information about 15
  notifications 34–35
  operating system 22
  pipes 35
  ports 35, 38
  sockets 35, 38
bsdmake tool 144
bugs, reporting 14
Build Applet 140
built-in commands 109
bundles 37, 73

C

C development 56
C++ development 56
c2ph tool 147
CalendarStore.framework 60, 116
Carbon application environment 56–57
Carbon Event Manager 106
Carbon.framework 56, 122
CarbonCore.framework 123
CarbonSound.framework 122
cascading style sheets 67, 88
cat command 112
cd command 112
CD recording 61
CDSA 66
certificates
  and security 66
  storing in keychains 97
CFNetwork 102
CFNetwork.framework 123
CFRunLoop 35
CFSocket 35
CGI 88
ci tool 151
Classic environment
  overview 33
Clipboard Viewer 140
co tool 151
Cocoa.framework 56, 116
Cocoa
  and web services 88
  application environment 55–56
  Application Kit framework 115
  bindings 55
  Exception Handling framework 117
  Foundation framework 117
  text 47
code completion 127
Code Fragment Manager 32
code signing 33, 73, 99
Collaboration.framework 62, 116
collection objects 37
color management module 52
Color Picker 122
ColorSync 52
ColorSync.framework 121
command-line tools 89
Common Unix Printing System (CUPS) 52
CommonPanels.framework 122
compileHelp tool 149
contextual menu plug-ins 83
Core Animation 98
Core Audio 48–49, 83
Core Data 60
Core Foundation
  date support 38
  features 37

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

networking interfaces 123
when to use 100
Core Graphics 41
Core Image 45, 98
Core Image Fun House 138
Core Text 74, 98
Core Video 51
CoreAudio.framework 116
CoreAudioKit.framework 116
CoreData.framework 60, 116
CoreFoundation.framework 56, 116
CoreGraphics.framework 121
CoreMIDI.framework 116
CoreMIDIServer.framework 117
CoreServices.framework 56, 123
CoreText.framework 121
CoreVideo.framework 117
cp command 112
CPlusTest.framework 124
CpMac tool 152
CrashReporterPrefs 140
cscope tool 147
csh shell 91
CSS. *See* cascading style sheets
ctags tool 147
CUPS 52
current directory 110
cvs tool 151
cvs-unwrap tool 151
cvs-wrap tool 151

D

daemons 90
Darwin 17, 21–24
Dashboard 77
Dashboard widgets 85
Dashcode 134
data corruption, and shared memory 36
data formatters 128
data model management 60
data synchronization 67
databases 67
date command 112
debug file formats 31
debugging 130
defaults tool 146
deprecated APIs, finding 96
DeRez tool 150
design principles
adaptability 100
attractive appearance 98
ease of use 97
interoperability 101
mobility 102
performance 95
reliability 99
technologies 99
use of modern APIs 96
developer tools, downloading 14
developer tools, overview 19
device drivers 22, 92–93
DHCP. *See* Dynamic Host Configuration Protocol
DictionaryServices.framework 123
diff tool 152
diff3 tool 152
diffpp tool 152
diffstat tool 152
digital paper 42
directory services 64
DirectoryService.framework 117
disc recording 61
DiscRecording.framework 117
DiscRecordingUI.framework 117
DiskArbitration.framework 117
Display Manager 106
distributed notifications 36–37
distributed objects 37
DNS daemon 90
DNS protocol 25, 64
Dock 77
Document Object Model (DOM) 68, 88
documentation
installed location 15
viewing 127
documents, opening 64
DOM. *See* Document Object Model
Domain Name Services. *See* DNS protocol
dprofpp tool 154
drag and drop 101
DrawSprocket.framework 117
DVComponentGlue.framework 117
DVDPlayback.framework 117
DVDs
playing 52
recording 61
DWARF debugging symbols 31
dyld 32
Dynamic Host Configuration Protocol (DHCP) 25

E

ease of use
and internationalization 97

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

explained 97
technologies for implementing 97
echo command 112
elegance, designing for 97
endian issues 30
enhancements, requesting 14
environment variables 113
environment.plist file 113
error tool 147
Ethernet 26
Event Manager 106
ExceptionHandling.framework 117
extcheck tool 157
Extensible Markup Language. *See* XML

F

fast user switching 75
FAT file system 24
Fax support 52
FIFO (first-in, first-out) special file 35
file system events 34, 100
file system journaling 23
file systems
support 23–24
File Transfer Protocol (FTP) 25
FileMerge 140
filename extensions 23, 105
files
browsing 76
long filenames 23
nib 107
opening 64
property list 74
quarantining 33
filters 83
find2perl tool 154
Finder application 76–77
FireWire
audio interfaces 117
device drivers 92
fix and continue 128
flex tool 155
flow control 111
Font Manager 106
Font window 122
ForceFeedback.framework 117
formatter objects 74
Foundation.framework 56, 117
fpr tool 145
frameworks 81–82, 115–124
FreeBSD 15, 17, 58
FSEvents API 34, 100
fsplit tool 145
FTP. *See* File Transfer Protocol
FWAUserLib.framework 117

G

gatherheaderdoc tool 149
gcc tool 144
gdb tool 146
genstrings tool 150
gestures 63
GetFileInfo tool 152
GIMP. *See* GNU Image Manipulation Program
GLUT.framework 117
GNU Image Manipulation Program (GIMP) 53
gnumake tool 144
gprof tool 148
graphics, overview 18

H

h2ph tool 154
h2xs tool 154
HALLab 137
handwriting recognition 63, 122
hardware architectures 29
headerdoc2HTML tool 149
heap tool 147
Help documentation 61
Help Indexer 140
Help Manager 105
Help.framework 122
HFS (Mac OS Standard format) 23
HFS+ (Mac OS Extended format) 23
HI Toolbox 61–62, 106, 122
HI Toolbox. *See* Human Interface Toolbox
HIObject 61
HIServices.framework 121
HIToolbox.framework 122
home directory 110
HotSpot Java virtual machine 39
HTML, editing 67
HTML
development 88
display 67
HTMLRendering.framework 122
HTMLView control 124
HTTP 25
HTTPS 25

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Human Interface Toolbox. *See* HI Toolbox
Hypertext Transport Protocol (HTTP) 25

I

I/O Kit 22
I/O Registry Explorer 140
`ibtool` tool 148
`ICADevices.framework` 118
ICC profiles 52
iChat presence 62
Icon Composer 140
icons 74
IDE. *See* integrated development environment
Identity Services 62
iDisk 76
`idlj` tool 156
`ifnames` tool 145
image effects 45
image units 83
`ImageCapture.framework` 122
`ImageIO.framework` 121
`ImageKit.framework` 123
images
  capturing 63
  supported formats 50
`indent` tool 145
`Info.plist` file 74
information property list files 74, 77
Ink services 63
`Ink.framework` 122
input method components 84
`InputMethodKit.framework` 63, 118
`install` tool 152
`install-info` tool 149
installation packages 73
`InstallerPlugins.framework` 118
`install_name_tool` tool 152
`InstantMessage.framework` 62, 118
Instruments 135
Instruments application 96
integrated development environment (IDE) 127
Interface Builder 107, 133
Interface Builder plug-ins 84
`InterfaceBuilderKit.framework` 124
internationalization 74
Internet Config 106
Internet support 24
interoperability
  explained 101
  technologies for implementing 101
interprocess communication (IPC) 34–37
`ioalloccount` tool 158
`IOBluetooth.framework` 118
`IOBluetoothUI.framework` 118
`ioclasscount` tool 158
`IOKit.framework` 118
`ioreg` tool 158
IP aliasing 27
IPSec protocol 25
IPv6 protocol 25
ISO 9660 format 23
iSync Plug-in Maker 141

J

`jam` tool 144
Jar Bundler 39, 138
`jar` tool 157
`jarsigner` tool 157
Java Native Interface (JNI) 39
Java Platform, Standard Edition/Java SE 57
`java` tool 156
Java Virtual Machine (JVM) 57
`java.awt` package 39
Java
  and web sites 88
  application environment 39, 57
`javac` tool 156
`javadoc` tool 156
`JavaEmbedding.framework` 118
`JavaFrameEmbedding.framework` 118
`javah` tool 156
JavaScript 88
`JavaVM.framework` 118
`javax.swing` package 39
JBoss 88
`jdb` tool 156
JIT (just-in-time) bytecode compiler 39
jumbo frame support 26

K

Kerberos 66
`Kerberos.framework` 118
kernel events 34
kernel extensions 92
kernel queues 34
`Kernel.framework` 118
kevents 34
`kextload` tool 157
`kextstat` tool 157

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

kextunload tool 157
Keychain Services 63–64, 66, 97
kHTML rendering engine 67
KJS library 67
kqueues 34

L

LangAnalysis.framework 121
language analysis 121
LatentSemanticMapping.framework 64, 118
launch items 90
Launch Services 64
LaunchServices.framework 123
ld tool 144
LDAP. *See* Lightweight Directory Access Protocol
LDAP.framework 118
leaks tool 147
LEAP authentication protocol 25
less command 112
lex tool 155
libtool tool 145
Lightweight Directory Access Protocol (LDAP) 25, 64
lipo tool 153
locale support 38
localization 74
lorder tool 145
ls command 112

M

Mac OS 9 migration 105–106
Mac OS Extended format (HFS+) 23
Mac OS Standard format (HFS) 23
Mach 21–22
Mach messages 37
Mach-O file format 31
Macromedia Flash 88
make tool 144
MallocDebug 139
malloc_history tool 147
man pages 109
Mandatory Access Control (MAC) 33
MDS authentication protocol 25
MediaBrowser.framework 122
memory
  protected 21
  shared 36
  virtual 21
merge tool 152
MergePef tool 153
Message.framework 119
metadata importers 84
metadata technology 72
Microsoft Active Directory 64
MIDI
  frameworks 116
mkbom tool 153
mkdep tool 144
mkdir command 112
MLTE. *See* Multilingual Text Engine (MLTE)
mobility
  explained 102
  technologies for implementing 102
modern APIs, finding 96
more command 112
Mouse keys. *See* accessibility
MS-DOS 24
multihoming 27
Multilingual Text Engine (MLTE)
  overview 48
Multiple Document Interface 104
multitasking 21
mv command 112
MvMac tool 153

N

Name Binding Protocol (NBP) 25
named pipes 35
native2ascii tool 157
NavigationServices.framework 122
NBP. *See* Name Binding Protocol
NetBoot 27
NetBSD project 15
NetInfo 64
network diagnostics 28
network file protocols 24
Network File System (NFS) 24
Network Kernel Extensions (NKEs) 28
Network Lookup Panel 122
Network Time Protocol (NTP) 25
networking
  features 24–28
  file protocols 24
  routing 27
  supported protocols 24
NFS. *See* Network File System
nib files 107
nm tool 148
nmedit tool 145
notifications 36–37

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

NSOperation object 96
NSOperationQueue object 96
NT File System (NTFS) 23
NTFS 23
NTP. *See* Network Time Protocol

O

Objective-C 38, 55
Objective-C 2.0 38
Objective-C++ 55
open command 113
Open Directory 64–65
Open Panel 122
open tool 114
Open Transport 26, 106, 123
open-source development 16
OpenAL 49
OpenAL.framework 49, 119
OpenBSD project 15
OpenDarwin project 15
opendiff tool 152
OpenGL 44, 99, 115
OpenGL Driver Monitor 138
OpenGL Profiler 138
OpenGL Shader Builder 138
OpenGL Utility Toolkit 117
OpenGL.framework 119
OpenScripting.framework 122
osacompile tool 153
OSAKit.framework 119
osascript tool 153
OSServices.framework 123
otool tool 148

P

PackageMaker 142
packages 73
PacketLogger 140
pagestuff tool 148
PAP. *See* Printer Access Protocol
parent directory 110
password management 63
passwords, protecting 97
Pasteboard 101
patch tool 152
path characters 110
PATH environment variable 114
pathnames 110
pbprojectdump tool 144
PCI 22, 92
PCIe 22
PCSC.framework 119
PDF (Portable Document Format) 42, 53
PDF Kit 65
PDFKit.framework 123
PDFView 65
PEAP authentication protocol 25
pen-based input 84, 122
performance
  benefits of modern APIs 96
  choosing efficient technologies 95
  explained 95
  influencing factors 95
  technologies for implementing 95
  tools for measuring 146
Perl 88, 91
perl tool 153
perlbug tool 154
perlcc tool 153
perldoc tool 154
Personal Web Sharing 27
PHP 88, 91
pipes, BSD 35
Pixie 138
pl2pm tool 155
plug-ins 37, 82–85
plutil tool 146
PMC Index 139
pod2html tool 155
pod2latex tool 155
pod2man tool 155
pod2text tool 155
pod2usage tool 155
podchecker tool 155
podselect tool 156
Point-to-Point Protocol (PPP) 25
Point-to-Point Protocol over Ethernet (PPPoE) 25
pointers 103
porting
  from 32-bit architectures 103
  from Mac OS 9 105
  from Windows 104–105
ports, BSD 35, 38
POSIX 22, 29, 58
PostScript OpenType fonts 47
PostScript printing 53
PostScript Type 1 fonts 47
PowerPC G5 103
PPC Toolbox 105
PPP. *See* Point-to-Point Protocol
PPPoE. *See* Point-to-Point Protocol over Ethernet

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

predictive compilation 128
preemptive multitasking 21
preference panes 87–88
PreferencePanes.framework 119
preferences 38
Preferred Executable Format (PEF) 31, 32
print preview 53
Print.framework 122
PrintCore.framework 121
Printer Access Protocol (PAP) 25
printf tool 146
Printing Manager 105
printing
  dialogs 122
  overview 52
  spooling 52
project management 127
Property List Editor 74, 141
property list files 74
protected memory 21
pstruct tool 148
PubSub.framework 65, 119
pwd command 113
Python 91
python tool 154
Python.framework 119

Q

QD.framework 121
QTKit.framework 51, 119
Quartz 41–43, 98
Quartz Composer 136
Quartz Composer Visualizer 138
Quartz Compositor 43
Quartz Debug 138, 139
Quartz Extreme 42, 43
Quartz Services 41, 100, 102
Quartz.framework 123
QuartzComposer.framework 123, 124
QuartzCore.framework 45, 51, 119
Quick Look 71, 97
QuickDraw 46, 106, 121
QuickDraw 3D 105
QuickDraw GX 105
QuickDraw Text 106
QuickLook.framework 119
QuickTime 50–51
QuickTime Components 51, 85
QuickTime formats 50
QuickTime Kit 50, 51
QuickTime.framework 119

R

ranlib tool 145
raster printers 53
rcs tool 151
rcs-checkin tool 151
rcs2log tool 151
rcsclean tool 151
rcsdiff tool 151
rcsmerge tool 151
redo_prebinding tool 145
refactoring 132
reference library 15
Reggie SE 139
reliability
  explained 99
  technologies for implementing 99
  using existing technologies 99
Repeat After Me 141
Research Assistant 130
ResMerger tool 146
resolution independence 72
resolution independent UI 42
Resource Manager 106
Rez tool 150
RezWack tool 146
rm command 113
rmdir command 113
rmic tool 157
rmiregistry tool 157
Routing Information Protocol 27
RTP (Real-Time Transport Protocol) 50
RTSP (Real-Time Streaming Protocol) 50
Ruby 91
ruby tool 154
Ruby.framework 119
RubyCocoa.framework 119
run loop support 38
runtime environments 32

S

S/MIME. *See* Secure MIME
s2p tool 154
Safari plug-ins 85
sample code 15
sample tool 148
Saturn 140
Save Panel 122
scalar values, and 64-bit systems 103
schema 60
screen readers 76

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

screen savers 86–87
ScreenSaver.framework 119
script languages 90–91
Script Manager 106
scripting additions 91–92
scripting support 28
Scripting.framework 119
ScriptingBridge.framework 120
sdiff tool 152
Search Kit 65
SearchKit.framework 123
Secure MIME (S/MIME) 25, 66
secure shell (SSH) protocol 25
secure transport 66
security 33
  dialogs 122
  Kerberos 118
  Keychain Services 97
  overview 66
Security.framework 120
SecurityFoundation.framework 120
SecurityHI.framework 122
SecurityInterface.framework 120
sed tool 154
semaphores 36
Service Location Protocol 25
services 87, 101
SetFile tool 153
SFTP protocol 25, 66
sh shell 91
shared memory 36
sharing accounts 62
Shark 139
Shark application 96
shells
  aborting programs 113
  and environment variables 113
  built-in commands 109
  commands 112
  current directory 110
  default 109
  defined 109
  frequently used commands 112
  home directory 110
  parent directory 110
  redirecting I/O 111
  running programs 114
  specifying paths 110
  startup scripts 113
  terminating programs 112
  valid path characters 110
Sherlock channels 89
Shockwave 88
simg4 tool 149
simg5 tool 149
Simple Object Access Protocol (SOAP) 25, 58, 88
SLP. *See* Service Location Protocol
smart cards 119
SMB/CIFS 24
snapshots 132
SOAP. *See* Simple Object Access Protocol
sockets 35, 38
Sound Manager interfaces 122
source code management 131
source-code management 128
Spaces 75
speech recognition 66, 75
speech synthesis 66
SpeechRecognition.framework 122
SpeechSynthesis.framework 121
spelling checkers 84
Spin Control 139
SpindownHD 140
splain tool 155
SplitForks tool 153
spoken user interface 75
Spotlight importers 73, 84
Spotlight technology 72
SQLite 67
SRLanguageModeler 141
SSH protocol 25, 66
stabs debugging symbols 31
Standard File Package 105
startup items 90
stderr pipe 111
stdin pipe 111
stdout pipe 111
Sticky keys. *See* accessibility
streams 35, 38
strings 38
strings tool 148
svn tool 150
svnadmin tool 150
svndumpfilter tool 150
svnlook tool 150
svnserve tool 150
svnversion tool 150
Swing package 39
Sync Services 67
Syncrospector 141
SyncServices.framework 67, 120, 124
syntax coloring 127
System Configuration framework 100, 102
System.framework 120
SystemConfiguration.framework 120

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

T

Tcl 91
Tcl.framework 120
tclsh tool 154
TCP. *See* Transmission Control Protocol
tcsh shell 91
technologies, choosing 95
Terminal application 109
TextEdit 106
Thread Viewer 139
threads 28, 96
Time Machine 68, 79
time support 38
Tk.framework 120
TLS authentication protocol 25
Tomcat 88
tools, downloading 14
top tool 148
tops tool 146
Transmission Control Protocol (TCP) 25
transparency 42
Trash icon 77
TrueType fonts 47
trust services 66
TTLS authentication protocol 25
TWAIN.framework 120

U

UDF (Universal Disk Format) 23
UDP. *See* User Datagram Protocol
UFS (UNIX File System) 24
Unicode 74
unifdef tool 146
UnRezWack tool 146
update_prebinding tool 145
URL Access Manager 106
URLs
  opening 64
  support for 38
USB Prober 141
User Datagram Protocol (UDP) 25
user experience 71–74
user experience, overview 18

V

V-Twin engine 65
vecLib.framework 120, 121
Velocity Engine 42
Vertical Retrace Manager 105
video effects 51
video formats 50
vImage.framework 121
Virtual File System (VFS) 23
virtual memory 21
visual development environments 136
visual effects 45
vmmap tool 147
vm_stat tool 147
VoiceOver 75
volumes 110

W

weak linking 32
Web Kit 67–68, 100
web services 68, 88
web streaming formats 50
WebCore.framework 124
WebDAV 24
WebKit.framework 124
WebObjects 58, 88
websites 88
window layouts 74
window management 43
workflow, managing 83
WSDL 58

X

X11 environment 29, 58
Xcode 127
Xcode Tools, downloading 14
xcodebuild tool 144
XgridFoundation.framework 120
XHTML 88
XML-RPC 26, 88
XML
  and websites 88
  parsing 38, 68–69
  when to use 101
Xserve 89

Y

yacc tool 155

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.

Z

zero-configuration networking 27, 59
ZoneMonitor 139
zooming. *See* accessibility
`zsh` shell 91

2008-10-15 | © 2004, 2008 Apple Inc. All Rights Reserved.