Inside Mac OS X

UNIX Porting Guide

An Overview of How to Bring UNIX Applications to Mac OS X

May 2002

Apple Computer, Inc.
© 2002 Apple Computer, 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 Computer, 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 Computer, 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 book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for Apple-labeled or Apple-licensed computers

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

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

Apple, the Apple logo, AppleScript, Cocoa, ColorSync, Finder, Mac, Macintosh, and QuickTime are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Carbon and Quartz are trademarks of Apple Computer, Inc.

NeXT, NextStep, and OpenStep are trademarks of NeXT Software, Inc., registered in the U.S. 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.

PostScript is a trademark of Adobe Systems Incorporated.

Simultaneously published in the United States and Canada

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, 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 MANUAL, 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

Chapter 1 **Introduction** 7

Who Should Read This Document? 8
How to Use This Document 8
Finding More Information 9

Chapter 2 **What You Need To Know About Mac OS X** 11

The Family Tree 11
  BSD 11
  Mach 12
  NEXTSTEP 12
  Earlier Version of the Mac OS 12
Mac OS X and Darwin 13
What Macintosh Users Expect 14
  Benefits 14
  Responsibilities 15

Chapter 3 **The Basic Port** 17

Preparation 17
  Installing Open Source Development Tools 17
  The Mac OS X Developer Tools 18
  Building Legacy Projects With Project Builder 19
  Windowing Environment Considerations 20
Compiling Your Code 21
  GNU Autoconf 21
  Compiler Flags 21
  Executable Format 23
  Dynamic Libraries and Plug-ins 23
  Bundles 24
    Application Bundles 24

© Apple Computer, Inc. May 2002

Frameworks 24

Chapter 4 Porting the User Interface 25

Choosing a Graphical Environment 25
What Kind of Application Are You Porting? 25
How Well Does It Need to Integrate With Mac OS X? 26
Does Your Application Require Cross-Platform Functionality? 26
Cocoa, Java, and Carbon 28
Cocoa 29
Benefits of Cocoa Development 29
Drawbacks of Cocoa Development 29
Example: Calling C or C++ Code With Cocoa 30
Java 33
Benefits of Java Development 34
Drawbacks of Java Development 34
Carbon 34
Benefits of Carbon Development 34
Drawbacks of Carbon Development 34
Lower-Level Graphics Technologies 35
Quartz 35
Benefits of using Quartz 36
Drawbacks to using Quartz 36
OpenGL 36
Benefits of using OpenGL 36
Drawbacks to using OpenGL 37
QuickTime 37
Benefits of using QuickTime 37
Drawbacks to using QuickTime 37
Traditional UNIX Graphical Environments 37
X11R6 38
Benefits of X11R6 Development 38
Drawbacks of X11R6 Development 38
Tcl/Tk 38
Benefits of Tk Development 39
Drawbacks of Tk Development 39
Qt 39

© Apple Computer, Inc. May 2002

Benefits of Qt Development 39
Drawbacks in Qt Development 39

Chapter 5 Additional Features 41

Audio Architecture 41
Boot Sequence 42
Configuration Files 44
Device Drivers 44
The File System 45
File-System Structure 45
Supported File-System Types 46
The Kernel 46
NetInfo 47
Example: Adding a User From the Command-Line 48
Role-Based Authentication 49
Scripting Languages 50
Security Services 51

Chapter 6 Distributing Your Application 53

Package It 53
Disk Copy 53
Tell the World About It 56

Glossary 57

Index 59

© Apple Computer, Inc. May 2002

© Apple Computer, Inc. May 2002

Introduction

Mac OS X is a modern operating system that combines the power and stability of UNIX-based operating systems with the simplicity and elegance of the Macintosh. For years, power users and developers have recognized the strengths of UNIX and its offshoots. While UNIX-based operating systems are indispensable to developers and power users, consumers have rarely been able to enjoy their benefits because of the perceived complexity. Instead consumers have lived with a generation of desktop computers that could only hope to achieve the strengths that UNIX-based operating systems have had from the beginning.

The introduction of UNIX-like operating systems such as FreeBSD and GNU/Linux for personal computers was a great step in bringing the power and stability of UNIX to the mass market. Generally though, these projects were driven by power users and developers for their own use, without making design decisions that would make UNIX palatable to consumers. Mac OS X on the other hand, was designed with end users in mind from the beginning. With this operating system, Apple builds its well-known strengths in simplicity and elegance of design on a UNIX-based foundation. Rather than reinventing what has already been done well, Apple is reconciling their strengths with the strengths brought about by many years of advancement by the UNIX community. Power, stability, simplicity, and elegance can all be spoken together in describing a single operating system. The time is ripe for Mac OS X as the next step in the evolution of UNIX-based operating systems.

This document helps to guide developers in bringing applications written for UNIX-based operating systems to Mac OS X. It provides the background needed to understand the operating system. It touches on some of the design decisions, and it provides a listing and discussion of some of the main areas that you should be concerned with in bringing UNIX applications to Mac OS X. It also points out some of the advanced features of Mac OS X not available in traditional UNIX applications

© Apple Computer, Inc. May 2002

that you can add to your ported applications. This document is an overview, not a tutorial. In many regards it is a companion to the more extensive *Inside Mac OS X: System Overview*, but with a bias toward the UNIX developer.

Who Should Read This Document?

This document is designed to be read by developers bringing a UNIX application to Mac OS X. More specifically, it is targeted to UNIX developers who have not traditionally been Macintosh developers. It helps to answer broad questions about the platform as a whole as well as specific questions pertinent to bringing a UNIX application onto the Mac OS X platform. It assumes that you are comfortable with the details of programming from coding to using traditional UNIX development tools and techniques.

This document is not designed for pure Java developers. Mac OS X has a full and robust Java 2 Platform, Standard Edition (J2SE) implementation. If you have a pure Java application already it should just work in Mac OS X. More information on Java development can be found at http://developer.apple.com/techpubs/java/.

How to Use This Document

This document is a first stop for UNIX developers coming to Mac OS X. It contains many links to more extensive documentation. Specific details of implementation are covered here only in cases where it is not adequately covered in other places in the documentation set.

To use this document most effectively, first read Chapter 2, "What You Need To Know About Mac OS X" (page 11), to find out the basics about the Mac OS X platform and to get some of the high-level information you need to begin your port. If you already have an application that builds on other UNIX-based platforms, Chapter 3, "The Basic Port" (page 17), will help you find out how to compile your code on Mac OS X.

© Apple Computer, Inc. May 2002

This is where the majority of the work comes in on your side. You will need to make decisions concerning which, if any, Graphical User Interface to implement with your application. Chapter 4, "Porting the User Interface" (page 25), and more specifically "Choosing a Graphical Environment" (page 25) helps you with this.

If you want to refactor your application to take advantage of the rich feature set of Mac OS X, see Chapter 5, "Additional Features" (page 41), for examples of features available in Mac OS X.

Once you have a complete application, read Chapter 6, "Distributing Your Application" (page 53), for information on getting your application to your users.

Finding More Information

Developer documentation can be found at Apple's developer documentation website at http://developer.apple.com/techpubs/. This site contains reference, conceptual, and tutorial material for the many facets of development on Mac OS X. The Mac OS X Developer Tools CD includes a snapshot of the developer documentation, which is installed in /Developer/Documentation. The man(1) pages are also included with the The Mac OS X Developer Tools.

Apple Developer Connection (ADC) hosts a website full of information useful to UNIX developers targeting Mac OS X, http://developer.apple.com/unix. ADC also offers a variety of membership levels to help you in your development. These range from free memberships that give you access to developer software, to paid memberships that provide support incidents as well as the possibility of software seeds. More information on memberships is available at http://developer.apple.com/membership/.

Once a year in May, Apple hosts the Worldwide Developers Conference (WWDC) in San Jose, California. This is an extremely valuable resource for developers trying to get an overall picture of Mac OS X in general as well as specific implementation details of individual technologies. Information on WWDC is available on the ADC website.

© Apple Computer, Inc. May 2002

Apple hosts an extensive array of public mailing lists. These are available for public subscription and searching at http://lists.apple.com. The unix-porting list is highly recommended. The darwin-development and darwinos-users lists also offer much help but less specific to the task of porting.

In addition to Apple's own resources, many external resources exist. Two recommended websites are O'Reilly's Mac DevCenter, http://www.oreillynet.com/mac/, and http://www.stepwise.com.

© Apple Computer, Inc. May 2002

What You Need To Know About Mac OS X

The purpose of this chapter is to give you background information about Mac OS X. It starts with a brief discussion of its lineage and then explains the distinction between Darwin and Mac OS X. It concludes with a discussion of the benefits and responsibilities in bringing your applications to Mac OS X.

The Family Tree

Although this document covers the basic concepts in bringing UNIX applications to Mac OS X, it is by no means comprehensive. This section is provided to give you a hint on where to look for additional documentation by outlining how Mac OS X came to be. Knowing a little about the lineage of Mac OS X will help you to find more resources as the need arises.

BSD

Part of the history of Mac OS X goes back to Berkeley Software Distributions (BSD) UNIX of the early seventies. Specifically, it is based in part on BSD 4.4 Lite. On a system level, many of the design decisions are made to align with BSD-style UNIX systems. Many of the libraries are derived from NetBSD (http://www.netbsd.org/), while many of the utilities are from FreeBSD (http://www.freebsd.org/). For future development, Mac OS X has adopted FreeBSD as a reference code base for BSD technology. Work is ongoing to more closely synchronize all BSD tools and libraries with the FreeBSD-stable branch.

.

© Apple Computer, Inc. May 2002

Mach

Although Mac OS X must credit BSD for most of the underlying levels of the operating system, Mac OS X also owes a major debt to Mach. The kernel is heavily influenced in its design philosophy by Carnegie Mellon's Mach project. The kernel is not a pure microkernel implementation though since the address space is shared with BSD processes. For more information on the kernel and the distinctions between Mach and BSD, see "The Kernel" (page 46).

NEXTSTEP

In figuring out what makes Mac OS X tick, it is important to recognize the influences of NEXTSTEP and OPENSTEP in its lineage. Apple's acquisition of NeXT in 1997 was a major key in bringing Mac OS X from the drawing board into reality. Many of the parts of Mac OS X of interest to UNIX developers are enhancements and offshoots of the technology present in NEXTSTEP. From the "The File System" (page 45) to the "Executable Format" (page 23), and from the high-level "Cocoa" (page 29) APIs to "The Kernel" (page 46) itself, the lineage of Mac OS X as a descendant of NEXTSTEP is very evident.

Earlier Version of the Mac OS

Although it shares its name with earlier versions of the Mac OS, Mac OS X is a fundamentally new operating system. This does not mean that all that went before has been left out. Mac OS X still includes many of the features that Mac OS 9 and earlier had. Although your initial port to Mac OS X may not use any of the features inherited from the Mac OS 9, as you enhance the application, you might take advantage of some of the features provided by technologies like ColorSync or the Carbon APIs. Mac OS 9 is also the source of much of the terminology used in Mac OS X.

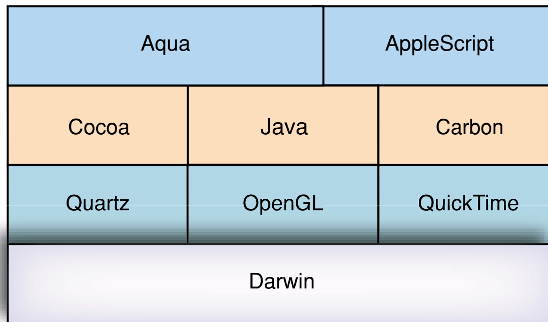
© Apple Computer, Inc. May 2002

Mac OS X and Darwin

The word Darwin is often used to refer to Mac OS X. In fact, in some circles Mac OS X itself is rarely mentioned at all. It is important to understand the distinction between the two–how they are related and how they differ.

Darwin is the core of the Mac OS X operating system. Although it could stand alone as an independent operating system, it includes only a subset of the features available in Mac OS X as a whole. Figure 2-1 shows how Darwin is related to Mac OS X as a whole.

Figure 2-1 Darwin's relation to Mac OS X



It is an open source project whose source code is available at http://www.opensource.apple.com/. This allows you as a developer access to the foundation of Mac OS X. Its openness also allows you to submit changes that you think should be reflected in Mac OS X as a whole. Darwin has been released as a separate project that runs on PowerPC-based Macintosh computers as well as x86-compatible computers. Although it could be considered a standalone operating system in its own right, many of the fundamental design decisions of Darwin are governed by its being embedded inside within Mac OS X. In bringing you applications to the platform, you should target Mac OS X version 10.1 (Darwin 5.1) or later.

© Apple Computer, Inc. May 2002

Mac OS X as a whole is not an open source project. As can be seen from Figure 2-1, there are many parts of Mac OS X that are not included in the open source Darwin components. This brings to light the need for you to decide where your application will fit into Mac OS X. You can, of course, simply port your application as a command-line tool or service, which is usually not that complicated. By doing this you gain a small benefit, in that it is now available to Mac OS X users. You will not be able to market to Mac OS X users as a whole though, since many users do not even know how to access the command line on their computers. It is important to remember that a port to the command line is just a first step to bringing your application to fully take advantage of Mac OS X. The next step would be to provide a graphical user interface (GUI). "Porting the User Interface" (page 25) gives you more information on how to decide which of the many available APIs to write to.

If you are adding a new graphical user interface to a command line application and want to take advantage of the greatest strengths of Mac OS X, you will probably want to use the Cocoa API's. In some cases, you may want to use a different API for reasons such as cross platform compatibility. If you decide to use a nonnative API, like X11R6, to provide a user interface for your Mac OS X application, it is important to remember that users and developers with a UNIX background might be perfectly content to just have the application running on Mac OS X. Traditional Macintosh users, however, will pass up an application with a traditional UNIX interface for a more integrated and modern interface. Whether you target a straight port to the Darwin layer or a more robust transformation of your application to take advantage of other Mac OS X technologies, like the Cocoa frameworks, is your decision.

What Macintosh Users Expect

In bringing your UNIX application to Mac OS X, you are entering a world where great emphasis is placed on user interactions. This brings many opportunities and benefits to you as a developer, but also some responsibilities.

Benefits

Macintosh users are known for their loyalty, but that loyalty is not blind. It is based on years of trust developed between Macintosh developers and the users. They are willing to spend their money on great applications because they know that Apple

© Apple Computer, Inc. May 2002

strives to give them a high-quality user environment. Apple developers are known for providing great applications for that environment. Bringing UNIX applications to Mac OS X can be very profitable if done correctly, both in pride and in business investment. Well-designed Macintosh applications of years past are the standards of today. PowerPoint, PhotoShop, Illustrator, and Excel are all applications that first made their name on the Macintosh. Now is the time to win the hearts of Macintosh users with the next great application. In a word, millions of possibly loyal users!

It is not all about business though. Apple has for years been known for its commitment to education. Mac OS X targets the education market for developers and is an ideal platform for learning for students. With its standards-based technologies as well as home-grown technologies, you have a platform ripe for use in educational application deployment and development.

Mac OS X also provides benefits in a development environment. Apple strives for standards first, then it adds that little bit that makes it better on a Mac. You have access to many of the development tools and environments that you have on other platforms, like Java, OpenGL, POSIX libraries, and the BSD TCP/IP stack, but you also have built-in benefits like the Apache Web server on every computer, the Cocoa object-oriented development environment, a PDF-based display system, Quartz, Kerberos compatibility, QuickTime, a dynamic core audio implementation, and a suite of world-class developer tools. By adding a native Mac OS X front end to your application, you can achieve a cost-effective new deployment platform with minimal additional development effort.

Mac OS X adds tremendous value both to you and your customers on a standards-based operating system.

Responsibilities

Along with benefits come responsibilities. If you have decided to make a full-featured Mac OS X application, here are a couple of simple but key guidelines to keep in mind.

A Mac OS X user should never have to resort to the command line to perform any task in an application with graphical user interface. This is especially important to remember since the BSD user environment may not even be installed on a user's system. The libraries and kernel environment are of course there by default, but the tools may not be.

© Apple Computer, Inc. May 2002

If you are making graphical design decisions, you need to become familiar with the *Inside Mac OS X: Aqua Human Interface Guidelines* that are available from the Apple developer website. These are the standards that Macintosh users expect their applications to live up to. Well-behaved applications from Apple and third-party developers give the Macintosh its reputation as the most usable interface on the planet.

The responsibilities boil down to striving for an excellent product from a users perspective. Mac OS X gives you the tools to make your applications shine.

© Apple Computer, Inc. May 2002

The Basic Port

A seasoned UNIX developer recognizes that no matter how similar two UNIX-based operating systems may be, there are always details that set one apart from another. This chapter highlights some of the key areas that you should be aware of when it comes to compiling your code base for Mac OS X. It notes details about compiler flags that are important to you, as well as giving you some insight into how to link different parts of your code in Mac OS X. Many of these topics are covered more extensively in other resources as noted.

Preparation

Before beginning bringing the basic port of your code to Mac OS X you need to make sure that you have the requisite tools for the task. You also need to be aware of what is available and isn't available to you by default.

Installing Open Source Development Tools

Because Mac OS X has a BSD core, you have access to the numerous open source tools, like the GNU tools, that you are already familiar with. Apple provides a basic selection of the most common tools with Mac OS X as an optional installation, so before using Mac OS X as a development platform you need to make sure that the BSD Subsystem is installed. You can check for this by looking for the BSD package receipt, `BSD.pkg`, in `/Library/Receipts`. If it is not present on your system or if you are installing Mac OS X for the first time, make sure that this package is installed:

1. Insert your Mac OS X CD.

© Apple Computer, Inc. May 2002

2. Double-click the Install Mac OS X application in the Welcome to Mac OS X folder.
3. In the window that opens, click the Restart button.
4. When your computer has restarted, follow the prompts until you get to the Installation Type phase. (This is indicated on the left side of the Install Mac OS X window.)
5. Click the Customize button.
6. Select the BSD Subsytem.
7. Proceed with the installation by following the prompts.

With the BSD subsystem installed, a look through `/bin` and `/usr/bin` should reveal a very familiar environment.

The Mac OS X Developer Tools package provides additional tools that you will need to install to round out your development environment. See "The Mac OS X Developer Tools" (page 18). These are not part of the default installation, but are essential to you since they contain some of the most important tools like the compiler (`gcc`) and debugger (`gdb`).

The Mac OS X Developer Tools

Apple provides a free, first class suite of Mac OS X–specific development tools with the operating system. Although they may not already be installed on your computer, they are usually distributed with the Mac OS X CDs. They are also available online from the Apple Developer Connection (ADC) Website, http://connect.apple.com. You will need an ADC account to download the Developer Tools. Free accounts are available for those who just need access to the tools. It is a good idea to check the ADC website for the most current version of the Mac OS X Developer Tools since updates are posted there.

After installing the Mac OS X Developer Tools you will have a selection of new tools for you to take advantage of:

- Inside `/Developer/Tools` are many Mac OS X–specific development tools. Documentation is provided in the form of man pages.

© Apple Computer, Inc. May 2002

- `/usr/bin` now contains more command-line tools than were supplied by the BSD package alone, most notably `cc` and `gdb`. `/usr/bin/cc` is gcc 2.9.52 in Mac OS X version 10.1, but gcc 3 is available from the Darwin CVS repository (http://www.opensource.apple.com/projects/darwin/).
- `/Developer/Applications` contains a wide assortment of graphical tools and utilities. Key among these are the following:

  Project Builder is a graphical integrated development environment for applications in multiple programming languages.

  Interface Builder provides a simple way to build complex user interfaces.

  FileMerge lets you graphically compare and merge files and directories.

  IORegistryExplorer helps you determine which devices are registered with the I/O KIT. See "Device Drivers" (page 44) for a discussion of the I/O Kit.

  MallocDebug analyzes all allocated memory in an application. It can measure the memory allocated since a given point in time and detect memory leaks

  PackageMaker builds easily distributable Mac OS X packages.

Documentation for these tools is available in the Developer Help Center in Project Builder's Help menu and online at http://developer.apple.com/techpubs/macosx/DeveloperTools/devtools.html.

Building Legacy Projects With Project Builder

Although Project Builder keeps track of build settings in its own preferences files for information beyond what could normally be maintained in a makefile, this does not mean that Project Builder cannot deal with your project's legacy makefiles. If you want to use Project Builder for development on Mac OS X, the following gives you a quick walk through of how to include a legacy makefile in a Project Builder project:

1. Open Project Builder.
2. Choose New Project from the File menu.
3. Select whatever project type you are targeting. If you ultimately want an application, select something like Cocoa Application. If you are just trying to build a command-line utility, select one of the tools, perhaps Standard Tool.

© Apple Computer, Inc. May 2002

4. Follow the prompts to name and save your project. A new default project of that type is opened.
5. From the Project menu, Choose New Target.
6. Select Legacy Makefile.
7. Follow the prompts to name that target. When you have done this, a target icon with the name you just gave it should show up in the Targets pane of the open Project Builder window.
8. Select that new target.
9. The information pane now changes to reflect the build information for this target. Modify these settings to include your makefile and any other settings you might need. For example, in the Custom Build Command pane, you could change Build Tool from `/usr/bin/gnumake` to `/usr/bin/bsdmake`. More information on the fields is available in Project Builder Help.
10. When you are ready to build the project, click the Build and Run button in the toolbar, select Build and Run from the Build menu, or just press Command-R.

This should at least get you started in bringing your application into the native build environment of Mac OS X.

Windowing Environment Considerations

Before you even attempt to compile an application in Mac OS X, you should be aware that the Mac OS X native windowing and display subsystem, Quartz, is based on the Portable Document Format (PDF). Quartz consists of a lightweight window server as well as a graphics rendering library for two-dimensional shapes. The window server features device-independent color and pixel depth, layered compositing, and buffered windows. The rendering model is PDF-based. Quartz is not an X Window System implementation. If you do need an X11R6 implementation, you can easily install one. If your application uses the X Window System you either need to port the user interface to a native Mac OS X environment or provide an X Window System implementation with your application. What to take into consideration in making this decision as well as information on the graphical environments available to you can be found in Chapter 4, “Porting the User Interface” (page 25),.

© Apple Computer, Inc. May 2002

Compiling Your Code

Now that you have the basic pieces in place, it is time to build your application. This section covers some of the more common issues that you may encounter in bringing your UNIX application to Mac OS X.

GNU Autoconf

If you are trying to bring a pre-existing command-line utility to Mac OS X that uses GNU Autoconf, you will find that most utilities just work; you just run `configure` and `make` as you would on any other UNIX-based system.

If Autoconf fails because it doesn't understand the architecture, try replacing the project's `config.sub` and `config.guess` files with those available in `/usr/libexec`. If you are distributing applications that use Autoconf, please include an up-to-date version of `config.sub` and `config.guess` so that Mac OS X users don't have to do anything to get your project to build.

You may also need to run `/usr/bin/autoconf` on your project before it works. Mac OS X includes Autoconf version 2.13 with the BSD tools. Beyond these basics, if the project does not build, you may need to modify your makefile according to some of the tips provided in the following sections. From that point, more extensive refactoring may be required.

Compiler Flags

When building your projects on Mac OS X, supplying or modifying the compiler flags of a few key options makes your job simple on most programs. These are usually specified by the `CFLAGS` argument in your makefile. The most common flags to append are these:

`-no-cpp-precomp`

Mac OS X uses precompiled headers to accelerate compiling C++ and Objective-C source code. By default this is done with the Mac OS X preprocessor and not the GNU C preprocessor. Since many open source

© Apple Computer, Inc. May 2002

projects are written with GNU C preprocessor extensions, which the Mac OS X preprocessor doesn't implement, you can turn the preprocessor off with the `-no-cpp-precomp` flag. This gives you the behavior of the GNU preprocessor. Many times if the error messages have anything to do with precompiled headers, this is the place to start.

> **Note:** In previous versions of Mac OS X, `-traditional-cpp` was suggested. Although this flag still works in the current version, the more accurate `-no-cpp-precomp` should be used instead.

`-shared`
: Use this flag to generate shared libraries on Mac OS X. Shared libraries on Mac OS X may be different from those you are accustomed to on other platforms. See "Dynamic Libraries and Plug-ins" (page 23).

`-flat_namespace`
: By default, Mac OS X builds its executables with a two-level namespace where dynamic libraries are resolved to a definition in a specific dynamic library when the image is built. If you need to depend on symbols that are implemented in your code and may also be defined elsewhere in the system, you need to use `-flat_namespace`. This flag helps you avoid symbol collisions where you and the system both define the same symbols; only one global symbol for each global symbol name is used by all images of the program. The `ld(1)` man page has a more detailed discussion of this flag. The two-level namespace is discussed at http://developer.apple.com/techpubs/macosx/ReleaseNotes/TwoLevelNamespaces.html.

`-bundle`
: Produces a Mach-O bundle format file, which is used to define symbols for an application. See the `ld(1)` man page for more discussion of this flag.

`-bundle_loader` *executable*
: Specifies an executable that loads the bundle output file being linked. Undefined symbols in that bundle are checked against the specified executable as if it were another dynamic library.

More extensive discussion for the compiler in general can be found at http://developer.apple.com/techpubs/macosx/ReleaseNotes/Compiler.html as well as the GNU C section of http://developer.apple.com/techpubs/macosx/DeveloperTools/devtools.html.

© Apple Computer, Inc. May 2002

Executable Format

The only executable format that the Mac OS X kernel understands is the Mach-O format. Some bridging tools are provided for classic Macintosh executable formats, but Mach-O is the native format. It is very different from the commonly used Executable and Linking Format (ELF). For more information on Mach-O, see *Inside Mac OS X: Mach-O Runtime Architecture*, available from http://developer.apple.com/techpubs/macosx/DeveloperTools/devtools.html.

Dynamic Libraries and Plug-ins

Mac OS X makes heavy use of dynamically linked code. The distinction between plug-ins and shared libraries in executable formats like ELF is not present in Mac OS X. In Mac OS X, they are treated the same.

Libraries that you are familiar with from other UNIX-based systems may not be installed in Mac OS X where they are on other systems. This difference is mainly due to the presence of a single dynamically loadable framework, `libSystem`, that contains the core system functionality. This single module provides the standard C runtime environment, input/output routines, math libraries, and most of the normal functionality required by command-line applications and network services. This module includes functions that you would normally expect to find in libc, libm, RPC services, resolver, and curses. `libSystem` is automatically linked against so you do not need to explicitly link against a library, for example curses, with the `-lm` flag. You do not need to worry about your code linking against a header whose implementation is in `libSystem` because stub files are already in place in the system. `libSystem` does not implement the threadsafe API variants.

It is important to note that Mac OS X does not support the concept of weak linking as it is found in systems like GNU/Linux. If you override one symbol, you must override all of the symbols in the object file.

If you need to dynamically load executable code with Mac OS X, use the `NSObjectFileImage` functions specified in `/usr/include/mach-o/dyld.h` and the `NSObjectFileImage(3)` man page. `NSObjectFileImage` provides the functionality you might be accustomed to finding in the `dl*` routines on other systems.

The `ld(1)` and `dyld(1)` man pages give more specific details of the dynamic linker’s implementation.

© Apple Computer, Inc. May 2002

Bundles

Bundles are used throughout the Mac OS X filesystem. Though you may not be familiar with them, they are not to be feared. Bundles are just directories that store executable code and the software resource related to that code in one discrete package. Bundles are discussed in more depth in *Inside Mac OS X: System Overview*. You may find that at times you need to link against an object file in a bundle with the `-bundle_loader` flag as specified in the `ld(1)` man page.

Application Bundles

Application bundles are special bundles that show up in the Finder as a single entity. Having only one item allows a user to double-click it to get the application with all of its supporting resources. If you are building Mac OS X applications, you should make application bundles. Project Builder builds them by default if you select one of the application project types. More information on application bundles is available in *Inside Mac OS X: System Overview*.

Frameworks

A framework is a type of bundle that packages a shared library with the resources that the library requires, including header files, images, and reference documentation. If you are trying to maintain cross-platform compatibility, you may not want to use them yourself, but you should be aware of them since you might need to link against them. For example, you might want to link against the Core Foundation framework. Since a framework is just one form of a bundle, you may do this by linking against `/System/Library/Frameworks/CoreFoundation.framework` with the `-bundle_loader` flag. A more thorough discussion of Frameworks is in *Inside Mac OS X: System Overview*.

© Apple Computer, Inc. May 2002

Porting the User Interface

Mac OS X Offers many options for transforming your applications with a graphical user interface from a UNIX code base to a native Mac OS X code base, or even for wrapping preexisting command-line tools or utilities with a graphical front end, making them available to users who never want to go to the command line.

Choosing a Graphical Environment

In choosing a graphical environment to use in bringing a UNIX-based application to Mac OS X, you will need to answer the questions posed in the following sections:

- "What Kind of Application Are You Porting?" (page 25)
- "How Well Does It Need to Integrate With Mac OS X?" (page 26)
- "Does Your Application Require Cross-Platform Functionality?" (page 26)

These questions should all be evaluated while weighing the costs and benefits of each environment. Mac OS X offers you a variety of cross-platform tool kits and APIs to work with, so it is important to pick the right one for your task.

What Kind of Application Are You Porting?

Are you bringing a preexisting code base to Mac OS X, or are you adding new functionality, like a graphical interface to a command-line application? Obviously if you already have a code base written to a particular API, and that API is supported on Mac OS X, you probably want to stick with that API for any large, complex application. For simple applications, or for applications where you are

© Apple Computer, Inc. May 2002

wrapping a command-line utility with a graphical user interface, you need to evaluate what API to use based on the information in "How Well Does It Need to Integrate With Mac OS X?" (page 26), and "Does Your Application Require Cross-Platform Functionality?" (page 26). It is important to recognize the benefits and drawbacks of each technology that are listed in the discussions on the individual technologies.

How Well Does It Need to Integrate With Mac OS X?

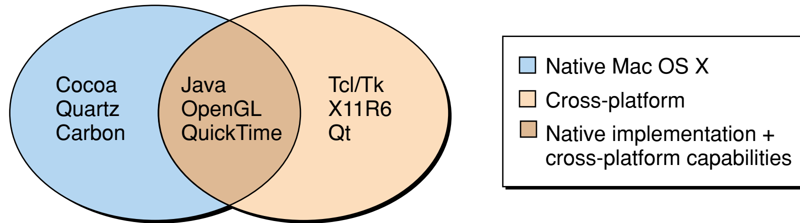
Who are you marketing your application to? If they are traditional UNIX users that just want to run a gene-sequencing application, for example, alongside of Microsoft Office, then it may be sufficient to just install an X Window System on their Mac OS X computer. You would simply port your X11R6-based application to Mac OS X, leaving your code as it stands aside from the little changes you may need to make to get it to compile in Mac OS X. If you sell that application, some customers might be happy to have it on their platform. However, if a competing product is released using Mac OS X native functionality, customers are likely to gravitate to that product. A hot topic in the science and technology industries is not only bringing a code base to Mac OS X, but then going the extra step to allow that code base to be accessed from the native user interface of Mac OS X. This is not a decision to be made trivially on large code bases, but it is one that can make or break a product in the market. The individual discussions of the APIs that follow should help you to make a well-informed decision.

Does Your Application Require Cross-Platform Functionality?

If you have an application that requires cross-platform functionality, Mac OS X has you set up for success. You have many options; some are built in and shipped with every version of the operating system—others require the installation of additional components. Figure 4-1 depicts the distinction between the cross-platform APIs that are native and those that aren't.

© Apple Computer, Inc. May 2002

Figure 4-1 Graphical environments



You can see that Mac OS X includes some of the standards in its native cross-platform APIs: Java, OpenGL, and QuickTime. There are also commercial and free implementations of some of the traditional UNIX technologies. If you are building a cross-platform application, you should evaluate which platforms you are targeting with your application and determine which API allows you to bring your UNIX-based application to Mac OS X. Table 4-1 lists the platforms on which Mac OS X cross-platform technologies run.

Table 4-1 Platforms of cross-platform technologies

API	Platforms
OpenGL	Mac OS X, UNIX-based systems, Windows
Java	Mac OS X, UNIX-based systems, Windows
QuickTime	Mac OS X, Windows
Qt	Mac OS X, UNIX-based X Windows Systems, Windows
Tcl/Tk	Mac OS X, UNIX-based systems, Windows
X11R6	Mac OS X, UNIX-based systems

Note: Although some of the technologies in Table 4-1 are supported in Mac OS 9, that is not considered here because your UNIX code base would not run in Mac OS 9.

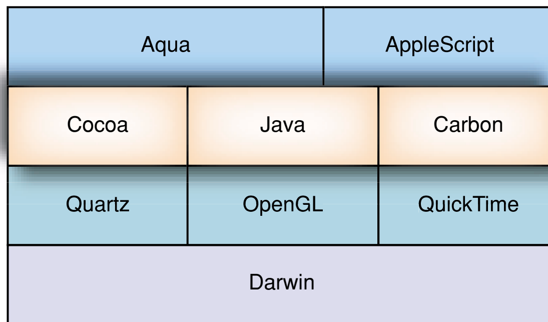
© Apple Computer, Inc. May 2002

In the following sections, brief descriptions of each of the technologies you could use for bringing your UNIX application to Mac OS X are discussed in a little more detail.

Cocoa, Java, and Carbon

Mac OS X includes three high-level native development environments that you can use for your application's graphical user interface: Cocoa, Java, and Carbon. These environments are full-featured development environments in their own right, and you can write complete applications in any one of these environments. In the context of this document, they are presented in light of using them as a front end for a UNIX-based back end. Writing to these environments enables you to build an application on top of your code base that is indistinguishable from any other native Mac OS X application. The Java or Cocoa frameworks are probably the environments that you will use in bringing UNIX applications to Mac OS X, although the Carbon frameworks are used by some developers. All three are outlined in the following sections.

Figure 4-2



© Apple Computer, Inc. May 2002

Cocoa

Cocoa is an object-oriented framework that incorporates many of Mac OS X's greatest strengths. It allows for rapid development and deployment with both its object-oriented design and integration with the Mac OS X development tools. Cocoa is divided into two major parts: Foundation and the Application Kit. Foundation provides the fundamental classes that define data types and collections; it also provides classes to access basic system information like dates and communication ports. The Application Kit builds on that by giving you the classes you need to implement graphical event-driven user interfaces.

The native language for Cocoa is Objective-C, which provides object-oriented extensions to standard C and Objective-C++. *Inside Mac OS X: The Objective-C Programming Language* describes the grammar of Objective-C and presents the concepts behind it. Objective-C is supported in gcc 2.95 and 3. Most of the Cocoa API is also accessible through Java.

Additional Cocoa information, including sample code, can be found at http://developer.apple.com/cocoa. Cocoa documentation including tutorials is available at http://developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html.

Benefits of Cocoa Development

- Rapid development environment
- Object-oriented framework design
- Excellent integration with Mac OS X developer tools
- Very robust feature set
- Can take advantage of existing C, C++, Objective-C, and Java code

Drawbacks of Cocoa Development

- No cross-platform deployment

© Apple Computer, Inc. May 2002

Example: Calling C or C++ Code With Cocoa

When designing an application from the ground up in Cocoa, you are in an object-oriented environment. You can also take advantage of Cocoa's object-oriented nature when converting preexisting code. You can use the Cocoa frameworks to wrap the functionality of C or C++ code.

The Objective-C language has been extended to understand C++. Often this is called Objective-C++, but the functionality remains basically the same. Because Cocoa understands Objective-C++, you can call native C and C++ code from Cocoa. This is one example of how you can take advantage of your code base while adding a Macintosh front end. An example is provided below of how Objective-C wraps together C, C++, and Objective-C++ functionality. Listing 4-1 shows the Objective-C main class.

Listing 4-1 main.m

```
#import <AppKit/AppKit.h>

int main(int argc, const char *argv[]) {
    return NSApplicationMain(argc, argv);
}
```

This gets everything started when the user double-clicks the application icon. A call is then sent to invoke a HelloController object by the NIB, a file that holds interface information. HelloController.m and HelloController.h follow.

Listing 4-2 HelloController.m

```
#import "HelloController.h"

@implementation HelloController

- (void)doAbout:(id)sender
{
    NSRunAlertPanel(@"About",@"Welcome to Hello World!",@"OK",NULL,NULL);
}

- (IBAction)switchMessage:(id)sender
```


© Apple Computer, Inc. May 2002

```
{
        int which=[sender selectedRow]+1;
        [helloButton setAction:NSSelectorFromString([NSString
stringWithFormat:@"%@%d:",@"message",which])];
}

- (void)awakeFromNib
{
    [[helloButton window] makeKeyAndOrderFront:self];
}

@end
```

Listing 4-3 `HelloController.h`

```
#import <AppKit/AppKit.h>

@interface HelloController : NSObject
{
    id helloButton;
    id messageRadio;
}
- (void)doAbout:(id)sender;
- (void)switchMessage:(id)sender;
@end
```

The communication between the C, C++, and the Objective-C code is handled as shown in Listing 4-4. The header file `SayHello.h` is shown in Listing 4-5.

Listing 4-4 `SayHello.mm`

```
#import "SayHello.h"
#include "FooClass.h"
#include <Carbon/Carbon.h>

@implementation SayHello
```


© Apple Computer, Inc. May 2002

```
- (void)message1:(id)sender
{
    NSRunAlertPanel(@"Regular Obj-C from Obj-C",@"Hello, World! This is a
regular old NSRunAlertPanel call in Cocoa!",@"OK",NULL,NULL);
}

- (void)message2:(id)sender
{
    int howMany;
    NSString *theAnswer;
    Foo* myCPlusPlusObj;
    myCPlusPlusObj=new Foo();
    howMany=myCPlusPlusObj->getVariable();
    delete myCPlusPlusObj;
    theAnswer=[NSString stringWithFormat:@"Hello, World! When our C++ object
is queried, it tells us that the number is %i!",howMany];
    NSRunAlertPanel(@"C++ from Obj-C",theAnswer,@"OK",NULL,NULL);
}

- (void)message3:(id)sender
{
    Alert(128,NULL); //This calls into Carbon
}

@end
```

Listing 4-5 SayHello.h

```
#import <AppKit/AppKit.h>

@interface SayHello : NSObject
{
}

- (void)message1:(id)sender;
- (void)message2:(id)sender;
- (void)message3:(id)sender;
@end
```


© Apple Computer, Inc. May 2002

The C++ class being wrapped by these Cocoa calls is shown in Listing 4-6. The header file, `FooClass.h`, is show shown in Listing 4-7.

Listing 4-6 FooClass.cpp

```
#include "FooClass.h"
Foo::Foo()
{
 variable=3;
}

int Foo::getVariable()
{
 return variable;
}
```

Listing 4-7 FooClass.h

```
class Foo {
public:
    Foo();
    int getVariable();
    void * objCObject;
private:
    int variable;
};
```

Java

Mac OS X includes a complete implementation of Java 2 Platform Standard Edition with the 1.3.1 JDK. Pure Java development on Mac OS works just as it would on other platforms. More information on Java development on Mac OS X can be found online at http://developer.apple.com/java.

© Apple Computer, Inc. May 2002

Benefits of Java Development

- Installed by default with Mac OS X
- Swing elements are displayed with the native Aqua look and feel of Mac OS X
- Cross-platform deployment

Drawbacks of Java Development

- Might not be as quick as a pure native implementation
- Complexities of communicating between two different languages if your code is C-based

Carbon

Carbon is an environment designed to bring existing Mac OS applications to Mac OS X. It is a very robust environment in its own right but was not designed initially to take full advantage of Mac OS X. You can bring your UNIX applications to Mac OS X using the Carbon API, but unless there is specific functionality you need in Carbon, you probably should avoid it.

Benefits of Carbon Development

- Well-documented feature set
- Integration with Mac OS X developer tools
- Very robust feature set
- Simple C and C++ integration

Drawbacks of Carbon Development

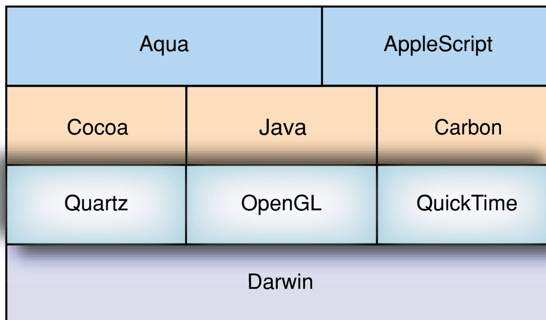
- No cross-platform deployment

© Apple Computer, Inc. May 2002

Lower-Level Graphics Technologies

With some code it is simple to abstract the display code from the underlying computational engine, but often this isn't the case. Especially for graphics-intensive applications, you may want to take advantage of directly calling the core graphic functionality of your targeted operating system. You still need to wrap this functionality in a higher-level API to display it to the screen and allow user interaction, but for times where you need to push pixels in very specific ways, Mac OS X offers you access to three first-class graphics technologies: Quartz, OpenGL, and QuickTime.

Figure 4-3 Low-level graphics technologies



Quartz

Quartz is the graphical system that forms the foundation of the imaging model for Mac OS X. Quartz gives you a standards-based drawing model and output format. Quartz provides both a two-dimensional drawing engine and the Mac OS X windowing environment. Its drawing engine leverages the Portable Document Format (PDF) drawing model to provide professional-strength drawing functionality. The windowing services provide low-level functionality like window

© Apple Computer, Inc. May 2002

buffering and event handling as well as translucency. Quartz is covered in more detail in *Inside Mac OS X: System Overview* and in Apple's Quartz website (http://developer.apple.com/quartz/).

Benefits of using Quartz

- PostScript-like drawing features
- PDF-based portability
- Included color management tools
- Unified print and imaging model

Drawbacks to using Quartz

- No cross-platform deployment

OpenGL

OpenGL is an industry-standard 2D and 3D graphics technology. It provides functionality for rendering, texture mapping, special effects, and other visualization functions. It is a fundamental part of Mac OS X and is implemented on many other platforms. Given its integration into many modern graphics chipsets and video cards, it is of special interest for programs that require intricate graphic manipulation. OpenGL's homepage gives more information on the technology in general at http://www.opengl.org. Apple's OpenGL page, at http://developer.apple.com/opengl/, gives more information on how it is integrated into Mac OS X.

Benefits of using OpenGL

- Cross-platform technology
- Native Mac OS X integration
- Included with every installation of Mac OS X
- Very robust feature set for handling graphics

© Apple Computer, Inc. May 2002

Drawbacks to using OpenGL

- Some level of integration at an application level is required

QuickTime

QuickTime is a powerful multimedia technology for manipulating, enhancing, storing, and delivering graphics, video, and sound. It is a cross-platform technology that provides delivery on Mac OS as well as Windows. More information on the technology can be found at http://developer.apple.com/quicktime/.

Benefits of using QuickTime

- Robust feature set for manipulating sound and video
- Cross-platform development

Drawbacks to using QuickTime

- Not supported on other UNIX-based systems

Traditional UNIX Graphical Environments

UNIX-based operating systems have grown to include many environments for providing a graphical interface to users. The X Window System is probably the most well known. As more specific needs have arisen, other architectures have been developed that assume an X Window implementation. Mac OS X does not use the X Window System by default, but it can. This means that X Window System based applications can be run, as can many of the alternative UNIX-style graphical environments. This section gives more details on some of these environments.

© Apple Computer, Inc. May 2002

X11R6

Mac OS X does not include an X11R6 implementation by default. If you are intent on only porting your application without adding any Mac OS X functionality, you can still run X-Windows on Mac OS X. Along with your application, you should either distribute an X11 implementation or at least instruct your users on how to download and configure one. There are commercial and free implementations of X Windows for Mac OS X.

XTools by Tenon Intersystems (http://www.tenon.com) and eXodus from Powerlan USA both provide X window servers that coexist with Aqua. If you do not need a commercial implementation of the X Window System, XFree86 offers a very robust free X11R6 implementation. The Mac OS X version of XFree86 is an active project with integral support from the XDarwin project. More information on the XDarwin project including tips for installing X Windows is available at http://www.xdarwin.org. XFree86 itself can be downloaded from http://xfree86.org/.

With an X Window System implementation, you are now free to use toolkits that you are already familiar with such as GTK or KDE.

Benefits of X11R6 Development

- The de-facto standard in display technology for UNIX-based operating systems
- Open source implementation is available

Drawbacks of X11R6 Development

- Complicated development environment
- Requires a large overhead of installed components
- Is not native to Mac OS X

Tcl/Tk

There is a native Aqua version of Tk available at http://tcl.sourceforge.net. With this you can easily add graphical elements to your existing Perl, Python, or Tcl scripts.

© Apple Computer, Inc. May 2002

Benefits of Tk Development

- Cross-platform development environment
- Easily integrates with Tcl and Perl scripts
- Open source implementation is available

Drawbacks of Tk Development

- Not supported by default in Mac OS X

Qt

Qt is a C++ toolkit for building graphical user interfaces. It is very popular in the UNIX world especially as it is used by the very popular K Desktop Environment, KDE. Trolltech has a native version of Qt, Qt/Mac, available for Mac OS X. If you already have a C++ application or are considering building one, Qt lets you build applications that run natively in Mac OS X as well as GNU/Linux and Windows. Information about Qt/Mac is available at http://www.trolltech.com. Qt/Mac does not run on top of X11 in Mac OS X, but the source code is compatible with Qt's X Windows implementation.

Benefits of Qt Development

- Cross-platform development environment
- Integrates easily with C++ code
- Robust feature set

Drawbacks in Qt Development

- May require licensing of third-party software

© Apple Computer, Inc. May 2002

© Apple Computer, Inc. May 2002

Additional Features

Although many parts of Mac OS X are the same as other UNIX-based operating systems, Mac OS X also includes many things that set it apart. This section highlights some of the key areas that you should be aware of. These may not be important for a basic port of a simple application, but the more robust your application and the more tightly it integrates with the underlying operating system, the more important it is to understand the additional functionality provided by the operating system. This section lists some of the key details that distinguish Mac OS X from most other UNIX-based operating systems. Most of the information here is covered only as an overview, with references to more detailed documentation where appropriate.

Audio Architecture

With Mac OS X, Apple has provided much of the audio functionality normally associated with third-party MIDI and other audio protocols right into the operating system itself. This gives developers a simple platform for developing dynamic audio applications. For end users, it minimizes the configuration that normally is required to get high-end audio applications to work. As a UNIX developer, it means that if you have been looking for a platform to develop a robust audio application on, you now have one. Among the high-level features of the Mac OS X Core Audio subsystem are these:

- native multi-channel audio with plug-in support
- native MIDI
- audio Hardware Abstraction Layer (HAL)

© Apple Computer, Inc. May 2002

- a built-in USB class driver compliant with the USB audio specification
- a simplified driver model
- a direct relation with the I/O Kit through the `IOAudioDevice` class that enables rapid device-driver development

If you are developing applications that need access to the audio layer of Mac OS X, you can pursue the extensive resources available at http://developer.apple.com/audio/.

Boot Sequence

An examination of `/etc/rc` reveals Start System Services, where System Starter is called. This is an entry unfamiliar on most other UNIX-style operating systems. System Starter is an application that offers many benefits to developers who need to start something running at boot time. It starts up many of the services available in Mac OS X that are traditionally started in `/etc/rc`. In Mac OS X, `/etc/rc` gets the basics going and then System Starter allows for additional functionality. This allows multiple services to start in tandem, provides a more robust method for checking dependencies of services than just startup time, and even provides for localization of startup strings printed to the screen. When invoked, System Starter looks first at the contents of the `/System/Library/StartupItems` and then `/Library/StartupItems` for services to start. Within each of these directories you can see examples of services that are run by Mac OS X.

Each service to be run consists of a directory with at least two items:

- A shell script that contains the same types of commands traditionally seen in `/etc/rc`. When this script is run in the startup process is determined by the `StartupParameters.plist`. This file is named the same as the directory that contains it.
- `StartupParameters.plist` is an XML file with a simple key-value DTD. The property list determines which services will be started when, by looking at dependencies on other services. It also provides descriptions for the services and strings to print in the user interface as the system is starting.

© Apple Computer, Inc. May 2002

Optionally it contains a Resources directory in which you can include localizable strings for the messages printed to the screen.

For an example of how all this works, look at the contents of one of the included services such as Apache. In `/System/Library/StartupItems/Apache` you see Apache, Resources, and `StartupParamaters.plist`.

Apache contains a very straight forward shell script:

```
#!/bin/sh

##
# Start Web Server
##

. /etc/rc.common

if [ "${WEBSERVER:=-NO-}" = "-YES-" ]; then
    ConsoleMessage "Starting web server"

    apachectl start
fi
```

`StartupParamaters.plist`, as shown in Listing 5-1, has a bit more structure to it. It is a compact listing of properties that identify the particular item being started as well as what services it provides and what the prerequisites are to running it.

Listing 5-1 A startup item's `StartupParamaters.plist` file

```
{
  Description     = "Apache web server";
  Provides        = ("Web Server");
  Requires        = ("Disks", "Resolver");
  Uses            = ("NFS", "Network Time");
  OrderPreference = "None";
  Messages =
  {
    start = "Starting Apache web server";
    stop  = "Stopping Apache web server";
  };
```


© Apple Computer, Inc. May 2002

```
}
```

The resources is directory contains localizable strings that are sent to the screen. Mac OS X determines which language to use based on the default language set in System Preferences. It then looks at the `Localizable.strings` file in that language's folder to see if any of the strings in `StartupParameters.plist` are overridden for that language. If so, it displays the appropriate string instead of the default string.

Configuration Files

Often on a UNIX-based system you find system configuration files in /etc. This is still true in Mac OS X, but configuration information is also found in other places as well. Networking, printing, and user system configuration details are regulated by the NetInfo database by default. Applications usually make use of XML property lists (plists) to hold configuration information. You can view many example property lists by looking in ~/Library/Preferences.

It is important to keep in mind that if changing a configuration file in `/etc` does not have your desired effect, you should look to see if that information is regulated by information in the NetInfo database or is covered by an application's property list.

Device Drivers

Mac OS X implements an object-oriented programming model for developing device drivers. This technology is called the I/O Kit. It is a collection of system frameworks, libraries, tools, and other resources. This model is different from the model traditionally found on a BSD system. If your code needs to access any devices other than disks, you use the I/O Kit. I/O Kit programming is done with a restricted form of C++, embedded C++ (eC++), that omits features unsuitable for use within a multi threaded kernel. By modeling the hardware connected to a Mac OS X system and abstracting common functionality for devices in particular categories, the I/O

© Apple Computer, Inc. May 2002

Kit streamlines the process of device-driver development. I/O Kit information and documentation is available at http://developer.apple.com/techpubs/macosx/Darwin/IOKit/iokit.html.

The File System

The Mac OS X file system is similar to other UNIX-based operating systems, but there are some significant differences, which are described below.

File-System Structure

Basically the file-system structure of Mac OS X is similar to a BSD-style system. A quick glance at hier(7) should comfort you. When in doubt as to where to put things, you can put them where you would in a BSD-style system.There are some different directories that you might not recognize.

The default behavior of the Mac OS X Finder is to hide the directories that users normally would not be interested in, as well as invisible files like those preceded by a dot (`.`). This appearance is maintained by the Finder to promote simplicity in the user interface. As a developer, you might want to see the dot files and your complete directory layout. `/usr/bin/defaults` allows you to override the default behavior of hiding invisible files. To show all of the files that the Finder ordinarily hides, type in the following command in the shell:

```
defaults write com.apple.Finder AppleShowAllFiles true
```

Then restart the Finder either by logging out and back in or by choosing Force Quit from the Apple Menu.

There are a couple of other simple ways to view the contents of hidden folders without modifying the default behavior of the Finder itself. You can use the `/usr/bin/open` command or the Finder Go to Folder command. With `open(1)` you can open a directory in the Finder, hidden or not, from the shell. For example, `open /usr/include` opens the hidden folder in a new Finder window. If you are in the Finder and want to see the contents of an invisible folder hierarchy, choose Go to Folder from the Go menu, or just press command-~, and type in the pathname of your desired destination.

© Apple Computer, Inc. May 2002

For information on how to lay out the directory structure of your completed Mac OS X applications, consult *Inside Mac OS X: Aqua Human Interface Guidelines* and *Inside Mac OS X: System Overview*.

Supported File-System Types

Mac OS X supports Mac OS Extended (HFS+), the traditional Macintosh volume format, and the UNIX File System (UFS). HFS+ is recommended and is what most users have their system installed on. Some more server-centric installations have their system installed on UFS. If you develop on UFS, you should thoroughly test your code on an HFS+ system as well. On important thing to note about the HFS+ file system is that although it preserves case, it is not case sensitive. This means that if you have two files whose names differ only by case, the HFS+ file system regards them as the same file. This is rarely an issue, but it is something that you should be aware of. In designing your application, you should not attempt to put two objects with names that differ only by case in the same directory—for example Makefile and makefile.

The Kernel

The core of any operating system is its kernel. The Mac OS X kernel is also known as XNU. Though Mac OS X shares much of its underlying architecture with BSD, the kernel is one area where they differ significantly. XNU is based on the Mach microkernel design, but it also incorporates BSD features. It is not technically a microkernel implementation, but still has many of the benefits of a microkernel.

Why is it designed like this? Pure Mach allows you to run an operating system as a separate process on the system that allows for flexibility, but can also slow things down because of the translation between Mach and the layers above it. With Mac OS X, since the desired behavior of the operating system is known, BSD functionality has been incorporated in the kernel alongside Mach. The result is that the kernel combines the strengths of Mach with the strengths of BSD.

How does this relate to the actual tasks the kernel must accomplish? Figure 5-1 illustrates how the kernel's different personalities are manifested.

© Apple Computer, Inc. May 2002

Figure 5-1 XNU personalities

BSD	Mach
- Users and permissions - Networking stack - Virtual File System - POSIX	- Memory management - Messaging - I/O Kit

The Mach aspects of the kernel handle

- memory management
- Mach messaging and Mach inter process communication (IPC)
- device drivers

The BSD components

- manages users and permissions
- contains the networking stack
- provides a virtual file-system
- maintains the POSIX compatibility layer

See Inside *Mac OS X: Kernel Programming* for more information on why you would (or wouldn't) want to program in the kernel space, including a discussion on the kernel extension (KEXT) mechanism.

NetInfo

NetInfo is the built-in Mac OS X directory system that system and application processes can use to store and find administrative information about resources and users. NetInfo refers to the database that stores this information as well as the processes by which this information is fed to the system. By default each Mac OS X computer runs both client and server processes where the server only serves to the local client. You can also bind client computers to servers other than the local server.

© Apple Computer, Inc. May 2002

Information is then accessed in a hierarchical scheme where each client computer accesses the union of the information provided by first its local NetInfo server and then any higher-level NetInfo servers it is bound to.

NetInfo is important to be aware of because it is the default way that Mac OS X stores user and some network information. When a user is added, the system automatically adds their information to the NetInfo database. Traditional tools like `adduser` do not work as you might expect. You can add users in several ways:

- through the Users pane of System Preferences
- through `/Applications/Utilities/NetInfo Manager`
- from the command-line (see "Example: Adding a User From the Command-Line" (page 48))

More information on NetInfo can be found primarily in `netinfo(5)` and `lookupd(8)`. *Understanding and Using NetInfo* gives a broad overview. `netinfo(3)`, `netinfo(5)`, `nidump(8)`, `nicl(8)`, `nifind(1)`, `niload(8)`, `niutil(1)`, and `nireport(1)` round out details of implementation.

Example: Adding a User From the Command-Line

This section shows a simple example of using the command-line NetInfo tool `niutil` to add a user to the system. The example specifies some of the properties that you would normally associate with any user.

1. Create a new entry in the local (/) domain under the category /users.

   ```
   niutil -create / /users/username
   ```

2. Create and set the shell property to `bash`.

   ```
   niutil -createprop / /users/username shell /bin/bash
   ```

3. Create and set the user's full name.

   ```
   niutil -createprop / /users/username realname "User Name"
   ```

4. Create and set the user's ID.

   ```
   niutil -createprop / /users/username uid 503
   ```

5. Creates and sets the user's global ID property.

   ```
   niutil -createprop / /users/username gid 1000
   ```


© Apple Computer, Inc. May 2002

6. Create and set the username on the local domain as opposed to the network domain or another domain.

```
niutil -createprop / /users/username home /Local/Users/username
```

7. Make an entry for the password.

```
niutil -createprop / /users/username _shadow_passwd
```

8. Now set the password.

```
passwd username
```

9. To make that user useful, you might want to add them to the admin group.

```
niutil -appendprop / /groups/admin users username
```

This is essentially what System Preferences does when it makes a new user, but is presented here so you can understand a little bit more about what is going on behind the scenes with the NetInfo database. A look through the hierarchies in the NetInfo Manager application will also help you to understand how the database is organized.

Role-Based Authentication

By default there is no root user in Mac OS X. This is a deliberate design decision both for security and to simplify the user interface. Your applications should not assume that a user needs superuser access to the system. For power users and developers, `sudo` is provided to run privileged applications in the shell. Privileged applications can also be run by members of the admin group. By default, the admin group is included in the list of sudoers. You can assign users to the admin group in System Preferences:

1. Click the Users button.
2. Select a user from the list and click Edit User, or make a new user.
3. Click the Password tab.
4. Check Allow user to administer this computer.
5. That user can now use administrative applications as well as `sudo` in the shell.

© Apple Computer, Inc. May 2002

Although it is generally considered unsafe practice to log in as the root user, it is mentioned here since the root user is often used to install applications or in some development scenarios. If during development you need to enable root for yourself:

1. Launch `/Applications/Utilities/NetInfo Manager`.
2. Choose Domain > Security >Authenticate.
3. As prompted, enter your administrator name and password.
4. Now choose Domain > Security > Enable Root User. The first time you do this you need to select a root password.

Alternatively you can use `sudo passwd root` from the shell and set the appropriate root password.

Important
Do not assume that an end user can enable the root user. This information is provided to help you in development work only.

Scripting Languages

Mac OS X includes the ability to run shell scripts in its native `sh` compatible shell, `zsh` or the included `csh` and `tcsh`. You can also run Ruby, Perl, Python, or other scripts you have developed. In addition, Mac OS X provides an Apple-specific scripting language, AppleScript. Although AppleScript is immensely powerful and can be used to build applications itself, it is important to note that it is designed mainly to communicate with graphical components of the operating system. There are other uses you can find for it, but it is not a replacement for UNIX-style scripting languages. You can use it, though, to put a front end onto your traditional scripts.

AppleScript does conform to the Open Scripting Architecture (OSA) language. and can be used from the command line through the `osascript(1)` command. Other languages can be made OSA compliant enabling interaction with the operating system.

© Apple Computer, Inc. May 2002

Security Services

Mac OS X implements the Common Data Security Architecture (CDSA). If you need to use Authorization Services, Secure Transport, or certificates within the scope of CDSA, online documentation is available at http://developer.apple.com/techpubs/macosx/CoreTechnologies/ in the Security Services section.

© Apple Computer, Inc. May 2002

© Apple Computer, Inc. May 2002

Distributing Your Application

Developing an application is only part of the story. You must now get it out there for people to use. Given that this is a UNIX-based operating system, you could just tar and gzip your file, but this won't meet the needs of the general user. To complete the transition, you should package you application like other Mac OS X applications. This chapter walks you through some of those details since they are probably new to you as a UNIX developer.

Package It

The recommended form of application distribution is a compressed disk image. This allows for the preservation of resource forks that may be present, simple drag-and-drop installation, as well as the encryption of data if required. If your application is a single application bundle, you can simply put your application bundles along with any relevant documentation on a disk image with Disk Copy, compress it, and distribute it. If you have an application that requires administrator privileges to install into privileged directories or requires more than a simple drag-and-drop installation, you should use `/Developer/Applications/PackageMaker` to build installer packages for Apple's Installer application. The basics of using Disk Copy to make a disk image are given in "Disk Copy" (page 53). For help using PackageMaker, choose PackageMaker Help from the PackageMaker Help menu.

Disk Copy

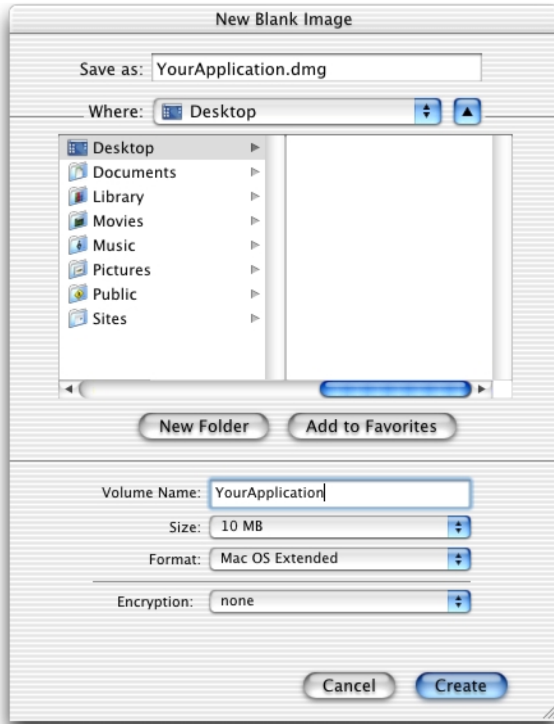
The following steps guide you through how to package your application as a disk image (`dmg` file) for distribution on Mac OS X.

© Apple Computer, Inc. May 2002

1. Open `/Applications/Utilities/Disk Copy` by double-clicking it.
2. From the Image menu, choose New Blank Image. Disk Copy opens a new window with customization options as in Figure 6-1 (page 55).
3. In the "Save as" text box, enter the name of the compressed file that you will distribute. By default a `.dmg` suffix is appended to the name you enter. Although it is not required, it is a good idea to retain this suffix for clarity and simplicity.
4. In the Volume Name text field, enter the name of the volume that you want to appear in the Finder of a user's computer. Usually this would be the same as the name of the compressed file without the `.dmg` suffix.
5. In the file browser, set the location to save the file on your computer. This has nothing to do with the installation location on the end user's computer, only where it saves it on your computer.
6. Set the Size pop-up menu to a size that is large enough to hold your application.
7. Usually you want to leave the Format set to Mac OS Extended (the HFS+ file format).
8. Leave Encryption set to none. If you change it the end user must enter a password before the image can be mounted, which is not the normal way to distribute an application.
9. Click Create.

© Apple Computer, Inc. May 2002

Figure 6-1 Disk Copy options



Once you have a disk image, mount it by double-clicking it. You can now copy your files to that mounted image. When you have everything on the image that you want, you should make your image read-only. Again from Disk Copy:

1. Choose Convert Image from the Image menu.
2. In the file browser, select the disk image you just modified and click Convert.
3. Choose a location to save the resulting file, change the image format to read-only, and click Convert.

You now have a disk image for your application that is easily distributable.

© Apple Computer, Inc. May 2002

Tell the World About It

Once you have an application, how do you get the word out? First, let Apple know about it. To get your application listed on Apple's main download page for Mac OS X, go to http://www.apple.com/downloads/macosx/submit/ and fill out the appropriate information about your application. You should also go to http://guide.apple.com/ and at the bottom of the page, click Submit a Product to get your application listed in the Apple Guide. You might then also want to send notices to http://www.versiontracker.com/and Macintosh news sites like http://www.maccentral.com/ and http://www.macnn.com/.

© Apple Computer, Inc. May 2002

Glossary

ADC See **Apple Developer Connection**

Apple Developer Connection The primary source for technical and business resources and information for anyone developing for Apple's software and hardware platforms anywhere in the world. It includes programs, products, and services and a website filled with up-to-date technical documentation for existing and emerging Apple technologies. The Apple Developer Connection is at http://www.apple.com/developer/.

Aqua The graphical user interface for Mac OS X.

bom (Bill Of Materials) A file in an installer package used by the Installer to determine which files to install, remove, or upgrade. It contains all the files within a directory, along with information about each file such as the file's permissions, its owner and group, size, its time of last modification, a checksum for each file, and information about hard links.

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.

Carbon An application environment for Mac OS X that features a set of 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, Mac OS 9, and all versions of Mac OS 8 later than Mac OS 8.1.

An advanced object-oriented development platform for Mac OS X. Cocoa is a set of frameworks with programming interfaces in both Java and Objective-C. It is based on the integration of OPENSTEP, Apple technologies, and Java.

Classic An application environment for Mac OS X that lets you run non-Carbon legacy Mac OS software. It supports programs built for both Power PC and 68K chip architectures and is fully integrated with the Finder and the other application environments.

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 Open Source technology.

© Apple Computer, Inc. May 2002

Darwin committer An individual who has been granted write access to Apple's core operating system CVS tree. Information on becoming a Darwin committer can be found at http://www.opensource.apple.com/.

dmg A Mac OS X disk image file.

Finder The system application that acts as the primary interface for file-system interaction.

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 folders themselves. HFS is a two-fork volume format.

HFS+ The Mac OS Extended file-system format. This file-system 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.

Mach-O The executable format of Mach object files. This is the default executable format in Mac OS X.

NetInfo Mac OS X's network administrative information database and information retrieval system. Many Mac OS X services consult the NetInfo database for their configuration information.

nib file An XML archive that describes the user interface of applications built with Interface Builder.

`.pkg` **file** A Mac OS X Installer file. May be grouped together into a meta package (`.mpkg`).

Plist See **Property List**.

Project Builder Apple's graphical integrated development environment. It is available free with the Mac OS X Developer Tools package.

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.

XNU The Mac OS X kernel. It combines functionality of Mach and BSD as well as the driver model, the I/O Kit.

© Apple Computer, Inc. May 2002

Index

Symbols

/etc/rc 42
/Library/StartupItems 42
/usr/bin/defaults 45

A

Apple Developer Connection (ADC) 9
AppleScript 50
Aqua Human Interface Guidelines 16, 46
audio drivers 42
audio Hardware Abstraction Layer (audio HAL) 41
autoconf 21

B

BSD
  device driver model 44
  file-system layout 45
  functionality in Mac OS X kernel 47
  relation to Mac OS X 11
-bundle_loadergcc flag 22, 24
-bundlegcc flag 22
bundles 24

C

Carbon
  as a development environment 34
  shared APIs with Mac OS 9 12
cc 19
Chapter 4, "Porting the User Interface" (page 25) 20
Cocoa
  available through Java 29
  calling C and C++ code from 30–34
  framework 29
Common Data Security Architecture (CDSA) 51
compiler
  included in Mac OS X Developer Tools 18
  version 19
cross-platform functionality 26–28
curses library 23
CVS, repository of opensource components 19

D

Darwin
  CVS repository 19
  mailing lists 10
  relation to Mac OS X 13–14
  XFree86 port 38
debugger, included in Mac OS X Developer Tools 18
developer tools 9, 15, 18
device drivers 44–45, 47
Disk Copy 53
disk image 53
documentation resources 9
dyld(1) 23
dynamically linked code 23

E

Embedded C++ (eC++) 44
Executable and Linking Format (ELF) 23

© Apple Computer, Inc. April 2002

F

FileMerge 19
-flat_namespace gcc flag 22
FreeBSD 11

G

gcc
  flags 21–22
  included in Mac OS X Developer Tools 18
  support of Objective-C 29
  version in Mac OS X 19
gdb, included in Mac OS X Developer Tools 18
GNU tools 17

H

HFS+ 46
hier(7) 45

I

I/O Kit 44–45
Interface Builder 19
IORegistryExplorer 19

J

Java 8, 33
  and the Cocoa API 29
  developer documentation 8
  implementation in Mac OS X 33–34
  included in Mac OS X 15

K

Kernel 46–47

L

ld(1) 22, 23, 24
libc 23
libm 23
libSystem 23

M

Mac OS 9, Mac OS X's relation to 12
Mac OS X
  Developer Tools CD 9
  relation to Darwin 13–14
Mac OS X build environment 19–20
Mach 12
  Executable Format 23
  interprocess communication (IPC) 47
  messaging 47
Mach-O 23
Macintosh user experience 15–16
mailing lists 10
makefiles, incorporating into native build environment 19–20
MallocDebug 19
memory management 47
MIDI 41

N

NetBSD 11
NetInfo
  man pages 48
  Manager 48
networking stack 47
NEXTSTEP 12
-no-cpp-precomp gcc flag 21

© Apple Computer, Inc. April 2002

`NSObjectFileImage` 23

O

Objective-C++ 30
Objective-C, support in gcc 29
Open Scripting Architecture (OSA) 50
OpenGL 36
OPENSTEP 12

P

PackageMaker 19
PDF based display system 15, 20, 35
Perl 50
POSIX 47
Project Builder 19
property lists 44
Python 50

Q

Qt 39
Quartz 20, 35
QuickTime 37

R

resolver 23
root user 49
RPC 23
Ruby 50

S

Security Services 51
`-shared`gcc flag 22
shell scripting 50

T

Tcl/Tk 38

U

UFS 46

V

virtual file-system 47

W

weak linking 23
Worldwide Developers Conference 9

X

X Window System 20, 26, 38
X11R6 20, 38
XDarwin 38
XNU 46

© Apple Computer, Inc. April 2002

© Apple Computer, Inc. April 2002