

- [Table of Contents](#)
- [Index](#)

Mac OS® X Advanced Development Techniques

By [Joe Zobkiw](#)

START READING

Publisher: Sams Publishing

Pub Date: April 22, 2003

ISBN: 0-672-32526-8

Pages: 456

Mac OS X Advanced Development Techniques introduces intermediate to advanced developers to a wide range of topics they will not find so extensively detailed anywhere else.

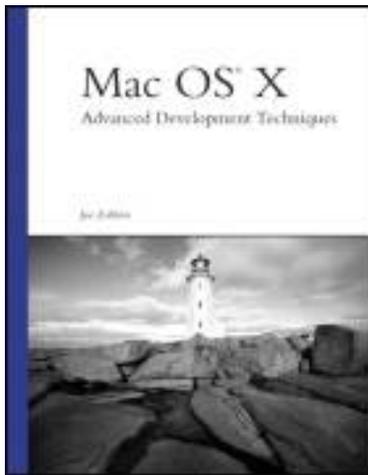
The book concentrates on teaching Cocoa development first, and then takes that knowledge and teaches in-depth, advanced Mac OS X development through detailed examples. Topics covered include: writing applications in Cocoa, supporting plug-in architectures, using shell scripts as startup items, understanding property lists, writing screen savers, implementing preference panes and storing global user preferences, custom color pickers, components, core and non-core services, foundations, frameworks, bundles, tools, applications and more. Source code in Objective-C, Perl, Java, shell script, and other languages are included as appropriate.

These solutions are necessary when developing Mac OS X software, but many times are overlooked due to their complexities and lack of documentation and examples. The project-oriented approach of *Mac OS X Advanced Development Techniques* lends itself perfectly to those developers who need to learn a specific aspect of this new OS. Stand-alone examples allow them to strike a specific topic with surgical precision. Each chapter will be filled with snippets of deep, technical information that is difficult or impossible to find anywhere else.

Brought to You by

Team LiB

Like the book? Buy it!



- [Table of Contents](#)
- [Index](#)

Mac OS® X Advanced Development Techniques

By [Joe Zobkiw](#)

START READING

Publisher: Sams Publishing

Pub Date: April 22, 2003

ISBN: 0-672-32526-8

Pages: 456

[Copyright](#)

[Preface](#)

[About the Author](#)

[Acknowledgments](#)

[We Want to Hear from You!](#)

[Reader Services](#)

[Foreword](#)

[Introduction](#)

[Part I: Overview](#)

[Chapter 1. Introduction to Mac OS X](#)

[Darwin](#)

[UNIX](#)

[File Systems](#)

[Industry Standard Protocols](#)

[Quartz](#)

[OpenGL](#)

[QuickTime](#)

[Classic \(OS 9\)](#)

[Carbon](#)

[Cocoa](#)

[Java](#)

[Aqua](#)

[Software](#)

[Conclusion](#)

[Chapter 2. Introduction to Programming in Mac OS X](#)

[Programming Languages](#)

[Tools](#)

[Frameworks](#)

[Conclusion](#)

[Part II: Application Level](#)

[Chapter 3. Cocoa Applications](#)

[The Result](#)

[The Project](#)

[Try This](#)

[Conclusion](#)

[Chapter 4. Cocoa Plug-ins](#)

[The Results](#)

[The First Project: plistPlugin](#)

[SearchWindowController](#)

[The Second Project: MyNSBP_App and MyNSBP_Plugin](#)

[A Word About Symbol Collisions](#)

[Try This](#)

[Conclusion](#)

[Chapter 5. Carbon Plug-ins](#)

[The Results](#)

[The CFPlugin Project](#)

[What Is a UUID?](#)

[The CFPlugin Project Source Code](#)

[The CFPluginCarbonApp Host Application Project](#)

[The CFPluginCocoaApp Host Application Project](#)

[Try This](#)

[Conclusion](#)

[Chapter 6. Frameworks](#)

[Apple Frameworks](#)

[Third-Party Frameworks](#)

[The Result](#)

[The Carbon Framework Project](#)

[The Cocoa Framework Project](#)

[The Cocoa Application Project](#)

[Try This](#)

[Conclusion](#)

[Part III: Enhancing the System](#)

[Chapter 7. System Services](#)

[The Result](#)

[The Project](#)

[The Application Connection](#)

[Try This](#)

[Conclusion](#)

[Chapter 8. System Preference Panes](#)

[The Result](#)

[The Project](#)

[More NSPreferencePane](#)

[Try This](#)

[Conclusion](#)

[Chapter 9. Status Items](#)

[The Result](#)

[The Project](#)

[Try This](#)

[Conclusion](#)

[Chapter 10. Screen Effects](#)

[The Result](#)

[The Project](#)

[Try This](#)

[Conclusion](#)

[Chapter 11. Color Pickers](#)

[The Result](#)

[The Project](#)

[Try This](#)

[Conclusion](#)

[Part IV: Advanced Methods](#)

[Chapter 12. Threads](#)

[The Result](#)

[The Project](#)

[Serial Versus Parallel](#)

[Thread Priority](#)

[Thread Data](#)

[Try This](#)

[Conclusion](#)

[Chapter 13. Terminal](#)

[The Result](#)

[The Project](#)

[Try This](#)

[Conclusion](#)

[Chapter 14. XML-RPC](#)

[The Result](#)

[The PHP Server](#)

[The PHP Client](#)

[The Cocoa Client](#)

[Try This](#)

[Conclusion](#)

[Chapter 15. SOAP](#)

[The Result](#)

[The PHP Server](#)

[The PHP Client](#)

[The Cocoa Client](#)

[Try This](#)

[Conclusion](#)

[Chapter 16. Snippets](#)

[Calling Carbon from Cocoa](#)

[Drawing with QuickDraw in a Cocoa Application](#)

[Displaying the Username](#)

[Finding a User's Home Directory](#)

[Finding Application Support Directories](#)

[Display Alert Panel and Write to Log](#)

[Updating the Application's Dock Image](#)

[Updating the Application's Dock Menu](#)

[Open a URL](#)

[Get the Icon of a File](#)

[Scrolling About Box](#)

[Checking Network Status](#)

[Checking for Modifier Keys](#)

[Executing AppleScripts via URL](#)

[Defaults](#)

[Find](#)

[Sample](#)

[Localizing Your Cocoa Application](#)

[CURLHandle](#)

[Conclusion](#)

[Chapter 17. Conclusion](#)

[Part V: Appendixes](#)

[Appendix A. Source Code](#)

[Chapter 3—RadarWatcher](#)

[Chapter 4—MyNSBP_App](#)

[Chapter 4—MyNSBP_Desaturate](#)

[Chapter 4—MyNSBP_RemoveColor](#)

[Chapter 4—plistPlugin](#)

[Chapter 5—CFPlugin](#)

[Chapter 5—CFPluginCarbonApp](#)

[Chapter 5—CFPluginCocoaApp](#)

[Chapter 5—Shared](#)

[Chapter 6—MyCarbonFramework](#)

[Chapter 6—MyCocoaFramework](#)

[Chapter 6—MyCocoaApp](#)

[Chapter 7—MyTextService](#)

[Chapter 8—MyPreferencePane](#)

[Chapter 9—MyStatusItem](#)

[Chapter 10—MyScreenEffect](#)

[Chapter 11—MyColorPicker](#)

[Chapter 12—MyThread](#)

[Chapter 13—MyTerminal](#)

[Chapter 14—MyXMLRPC](#)

[Chapter 15—MySOAP](#)

[Appendix B. Online Resources](#)

[Index](#)

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Copyright

Copyright © 2003 by Sams Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

Library of Congress Catalog Card Number: 2002116065

Printed in the United States of America

First Printing: May 2003

Reprinted with corrections: June 2003

05 04 03 4 3 2

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the CD or programs accompanying it.

Credits

Acquisitions Editor

Katie Purdum

Development Editor

Scott Meyers

Managing Editor

Charlotte Clapp

Project Editor

Andy Beaster

Copy Editor

Rhonda Tinch-Mize

Indexer

Chris Barrick

Proofreader

Jody Larsen

Technical Editor

Mike Trent

Team Coordinator

Vanessa Evans

Media Developer

Dan Scherf

Designer

Gary Adair

Page Layout

Kelly Maish

Dedication

This book is dedicated to all my friends and family—with a special blessing to those we've lost.

[\[Team LiB \]](#)

Preface

[About the Author](#)

[Acknowledgments](#)

[We Want to Hear from You!](#)

[Reader Services](#)

[Foreword](#)

About the Author

Joe Zobkiw is president of TripleSoft Inc., a software development firm located in Raleigh, NC. He has been writing software for Macintosh, UNIX, and Windows operating systems since 1986. Joe has written numerous technical articles on software development-related topics throughout his career. His experience includes writing communication, utility, and business applications for commercial and private clients. This is his second book on advanced Macintosh software development.

Acknowledgments

This is the section where I thank all the people who made it possible to write the book you are now holding. Some of them introduced me to someone, some of them worked in the trenches, and some of them didn't complain while I coded for "just a few more minutes."

Love and most thanks to Catherine—Go Bucs!

Thanks also to Carole McClendon and everyone at Waterside Productions; Kathryn Purdum, Scott Meyers, and everyone at Sams Publishing; Mike Trent for an excellent technical review and foreword; Everyone on the various Cocoa and Mac OS X mailing lists, chat rooms, and Web sites for keeping me in the loop; Apple Computer for designing such great hardware and software and all the folks who work hard to put it together. Also thanks to Jeff Dopko, Bob Levitus, Matt Manlove, Marty Wachter, the Emmi family, the McNair family, and the Zobkiw family.

Special note to Marty: There is a hidden chapter in this book. If you read the seventh word of every third sentence, it will tell you how to make an iTunes plug-in.

Last, but certainly not least, thanks to you for making this purchase. I know that many books are vying for your earnings, and I'm glad you chose this one. I hope you enjoy it!

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone or email address. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@sampublishing.com

Mail: Mark Taber
Associate Publisher
Sams Publishing
201 West 103rd Street
Indianapolis, IN 46290 USA

Reader Services

For more information about this book or others from Sams Publishing, visit our Website at www.sampublishing.com. Type the ISBN (excluding hyphens) or the title of the book in the Search box to find the book you're looking for.

Foreword

If asked to define Mac OS X in 5 words or less, I would call it "a collection of new beginnings."

Some of these beginnings are obvious. With Mac OS X, Apple has delivered on its promise to bring a pre-emptive multi-tasking, memory-protected operating system to the Mac-using masses. Apple has also introduced a new user interface for Mac OS X, called Aqua, which is simple, inviting, and powerful. If you read through Apple's Web site, you will find many more examples, some even with color pictures.

But some beginnings are more profound. Many computer users are for the first time considering using an Apple computer, in part because of Mac OS X's power and ease of use. Many others are switching back to the Macintosh platform from other systems, drawn by Apple's innovative hardware and software efforts. Even IT managers are beginning to treat Mac OS X as a viable server platform.

When Mac OS X shipped in March, 2001, Apple began including developer tools with every copy—for free. This is a serious invitation to developers to explore Mac OS X from every angle: from writing applications using C, C++, Objective C, and Java; to writing web services using Perl, PHP, and Python. Mac OS X demystifies the previously complex, bringing shared libraries, plug-ins, and device drivers to the programming populace. For some, Cocoa represents the first time they really "got" object-oriented design. And there are those who view Mac OS X's POSIX API and tools, both from the UNIX world, as a major asset. In these ways and more Mac OS X offers something for every developer.

In the pages that follow, Joe Zobkiw will guide you through many of these stops on the Mac OS X road. Use these stops as departure-points when you begin your own projects. And once you feel comfortable, go off and explore the rest of Mac OS X's new beginnings on your own.

Michael Trent
Santa Clara, California
March 2003

Introduction

Welcome to *Mac OS X: Advanced Development Techniques!*

I hope you enjoy what I've put together for you here. The goal of this book is to create examples that you, as a developer, could use as jumping-off points for your own intermediate to advanced programming projects.

The book is being written for those of you who have had some OS X programming experience and understand basic programming concepts. Regarding experience, you should at least have created a "Hello World" application under OS X in either Cocoa or Carbon.

It is assumed that you will read the book more or less in order. Although each chapter can stand alone, similarities between projects are discussed in one chapter that might not be discussed in as much detail in later chapters. If you are unsure about anything, consult the list of online resources for help.

In addition, as I discuss a topic, I might not cover every possible aspect of it. My goal in writing this book is not to duplicate provided documentation but to show a working example of a particular project with enough insight to allow you to take it to the next level. Like any good teacher, I will not give you all the answers, but I will give you the tools to find them.

It is suggested that you explore the header files and Apple supplied documentation for any particular project. In fact, you might consider having your Web browser open to the online documentation so that you can look up unfamiliar classes that you come across in the book. The documentation provides cross-references to other classes and methods that can be very helpful as you learn the ins and outs of programming Mac OS X through my examples. The Cocoa Browser application, listed in [Appendix B](#), "Online Resources," is also a very useful tool to keep handy on your computer.

This book is filled with development examples. I attempt to show you how a technology works, yet not precisely how you should use it. Be sure to read up on Apple's Human Interface Guidelines before you dig in too deeply—especially if you are new to Mac OS X.

All sentences in the book could be prefaced with "As of this writing." As you know, especially in the world of software development, things can change quite rapidly. If something isn't working properly, check the release notes of your development software to see if something has changed. Also, keep an eye on this book's Web site for errata and downloadable source code. You can visit the Web site at <http://www.triplesoft.com/>.

So pull yourself up to the computer, crack your favorite beverage, and stay up late programming! I'll meet you when it's over.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Part I: Overview

[1 Introduction to Mac OS X](#)

[2 Introduction to Programming in Mac OS X](#)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

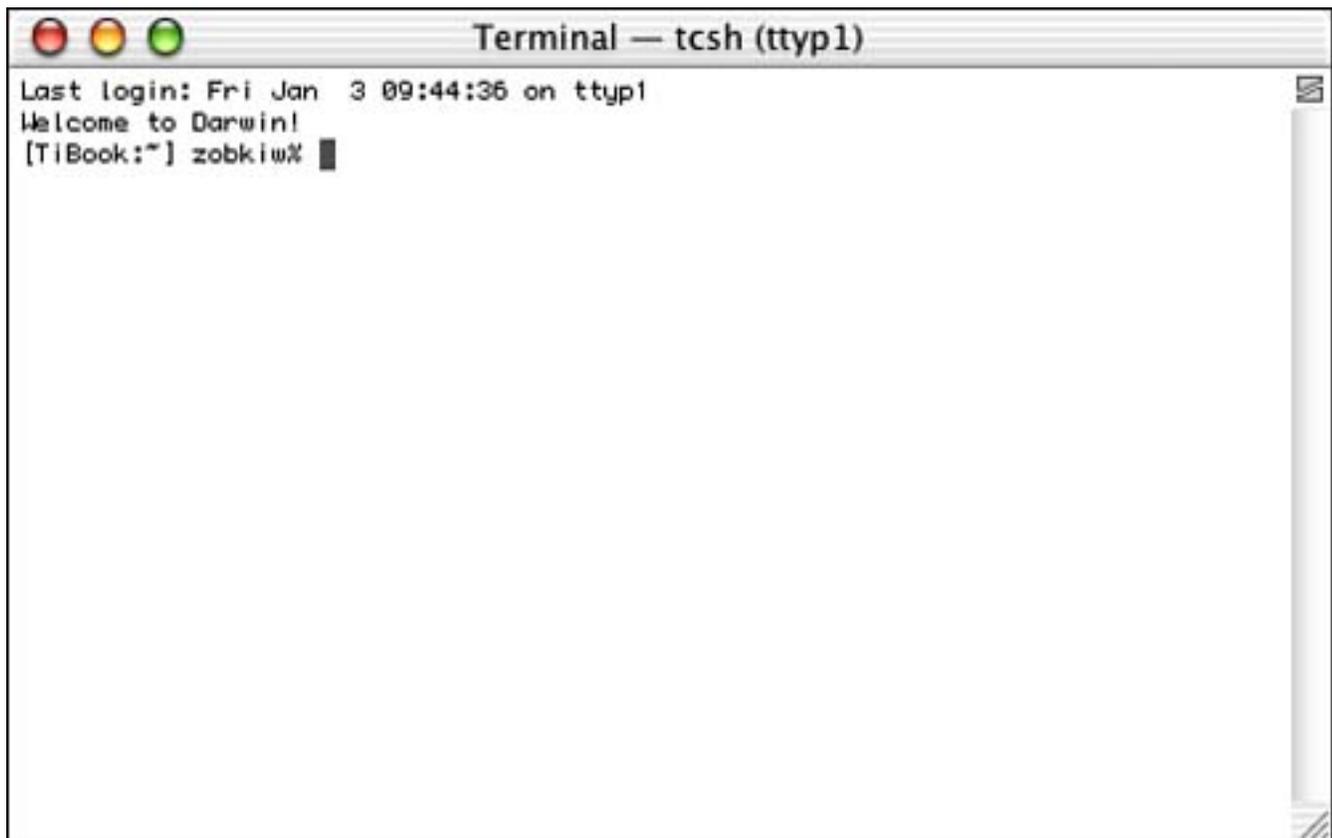
Chapter 1. Introduction to Mac OS X

"How do you describe Mac OS X? Batteries included!"

—Anonymous

Mac OS X is a Completely Redesigned implementation of the Macintosh operating system based on the BSD UNIX operating system. UNIX is an advanced multi-platform operating system that offers advanced features such as enhanced networking, high performance, and security. By focusing around the UNIX environment, the Macintosh now arguably becomes the most powerful consumer computer available today. It has so many built-in features now that you may well want for nothing more in an operating system. Simply, your Macintosh is still a Macintosh with the added benefit of being able to do anything that UNIX can do.

Figure 1.1. Terminal application in Mac OS X: Welcome to UNIX!



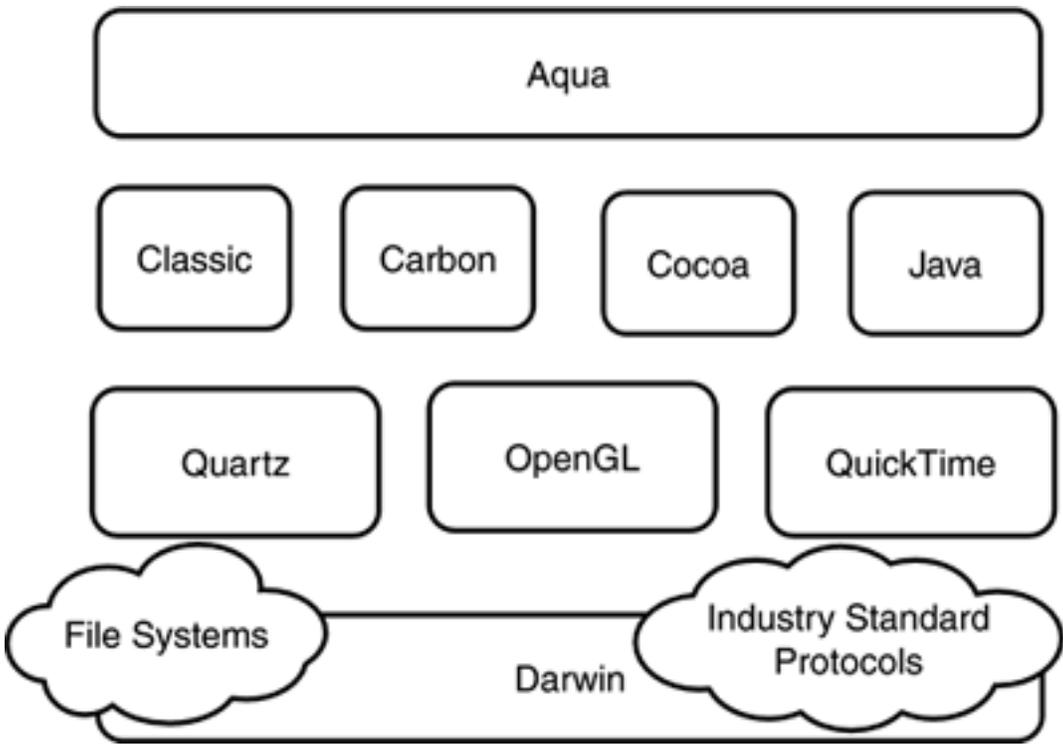
However, Apple didn't stop there. Not only is Mac OS X the most powerful Macintosh operating system to date, but it is also the most elegant by means of the Aqua user interface. Apple based the interface on the standard elements that Macintosh users expect and added many enhancements that make the user interface flow like nectar. Simply, the Aqua user interface is the most stylish ever created with its awesome three-dimensional widgets and drop shadows, as shown in [Figure 1.2](#).

Figure 1.2. Aqua.



Mac OS X contains many pieces that make up this incredible puzzle we call our operating system. Let's look at them briefly with a quick explanation and real-world examples of how you might take advantage of the power they offer. [Figure 1.3](#) shows how you might visualize the layers that make up Mac OS X.

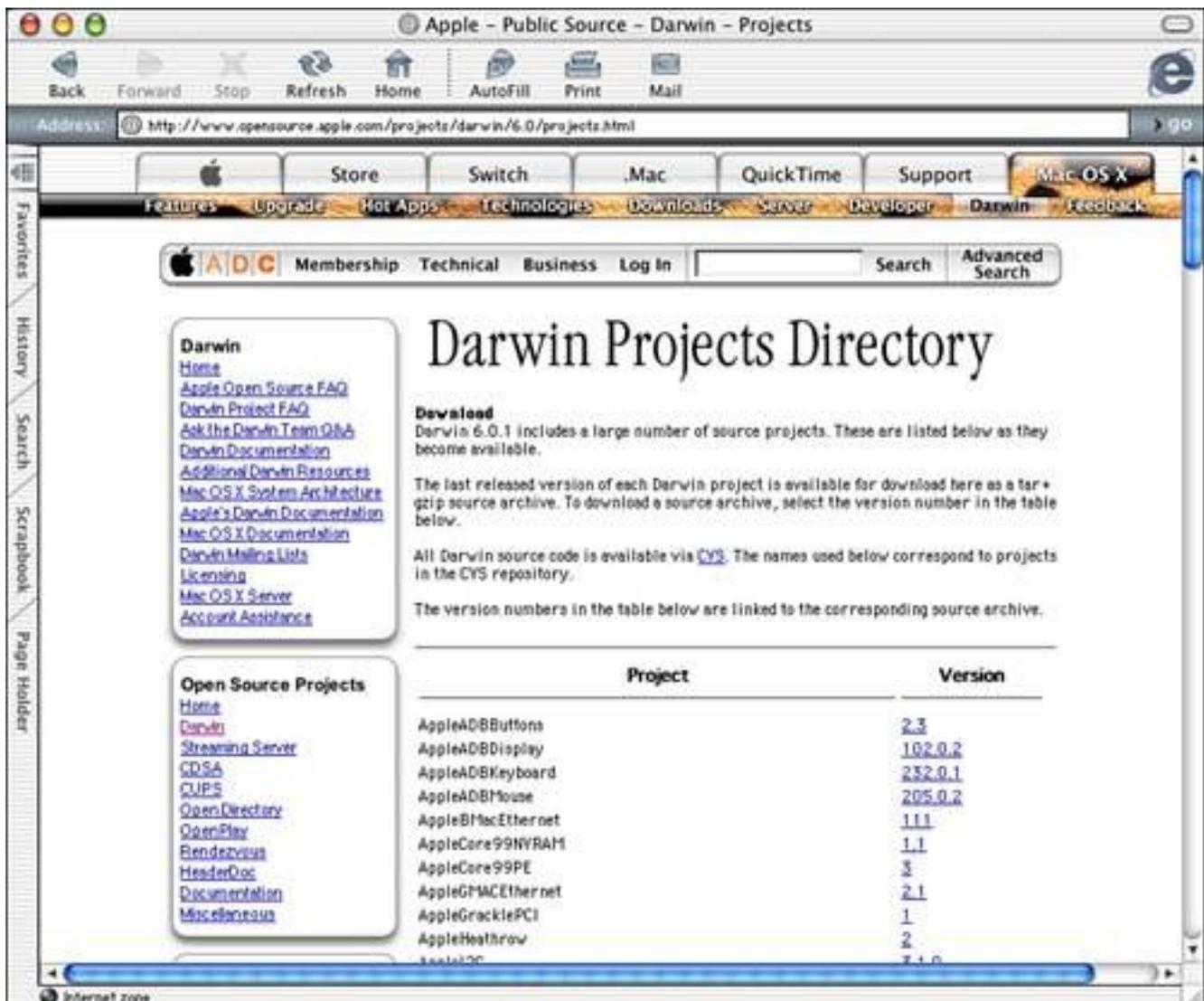
Figure 1.3. The many pieces of X.



Darwin

Mac OS X's core OS layer, named Darwin, is based on version 4.4 of the FreeBSD (Berkeley Software Distribution) operating system. It therefore includes all the power, flexibility, and stability of that proven UNIX implementation. If that isn't enough, the Darwin source code is available as a free download from Apple, as shown in [Figure 1.4](#). Apple also just recently released its own X11 implementation. It includes the full X11R6.6 technology, including a window server, libraries, and utilities.

Figure 1.4. Darwin Projects Directory where you can download all the Darwin source code.

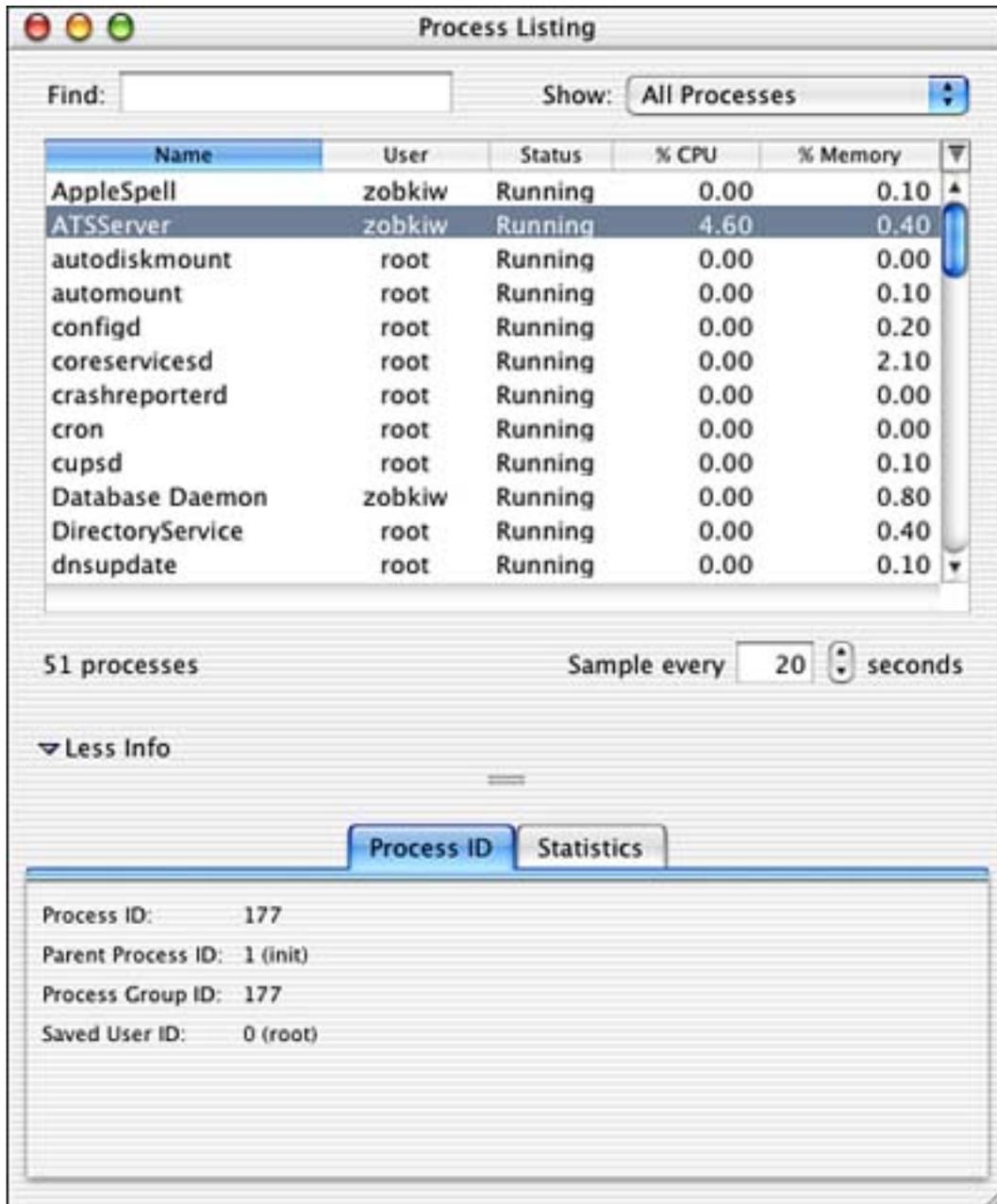


Because the Darwin source code is available, not only can you study it, but you can also change it. You can fix any obscure bugs you might find or add new features. Apple even has a Web form that allows you to submit your modifications for evaluation and possible inclusion in future releases.

UNIX

For those of you not familiar, UNIX is a powerful, multiuser operating system that has been in use in various capacities since the 1970s. It was primarily intended for multiuser computing on servers. Apple has found a delicate balance and has been able to hide much of UNIX's complexity for the novice user while still making it 100% configurable for the geek and developer (see [Figure 1.5](#)).

Figure 1.5. Process Viewer lists current processes and resource usage.



Many different versions of UNIX are available including AT&T, BSD, Linux, and others. Some are free, whereas some are not. Mac OS X descends from the BSD-flavor of UNIX, which dates back to the mid-70s. At that time, the University of California at Berkeley licensed UNIX from AT&T and began to create its custom version, which became known as BSD. Because the BSD source code was available to all, Apple

chose it as a base for Mac OS X.

If you've never used UNIX before, you are in for a real treat. UNIX servers are everywhere on the Internet and have a habit of acting like the Energizer Bunny; they just keep going and going. UNIX servers have some of the longest uptimes of any server on the Net. Given this penchant for few crashes and the fact that HTTP, FTP, TELNET, and SSH servers are all built into Mac OS X, you can be running and managing a high-powered network server in no time at all.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

File Systems

Mac OS X supports compatibility with a number of file systems and protocols including the original Macintosh HFS, HFS+, FTP, UFS, NFS, ISO 9660, MS-DOS, SMB, AFP, UDF, and WebDAV. Not only can Mac OS X mount all these file systems, but it can also boot from many of them. [Figure 1.6](#) shows Mac OS X connecting to a remote Windows File Sharing server.

Figure 1.6. Connecting to a Windows File Sharing server.



With the awesome support for numerous file systems, Mac OS X allows you to easily and quickly mount multiple file servers or shared directories on your desktop over the Internet. You can have your Windows machine at work mounted right next to your Mac iDisk and remote UNIX server—all at the same time. You can also make your own computer available to others as an SMB, FTP, or AFP server if you like—all at the same time. For developers, this is probably something you've been used to in the past in other environments; now it is finally a reality on the Macintosh without gobs of third-party programs.

[\[Team LiB \]](#)

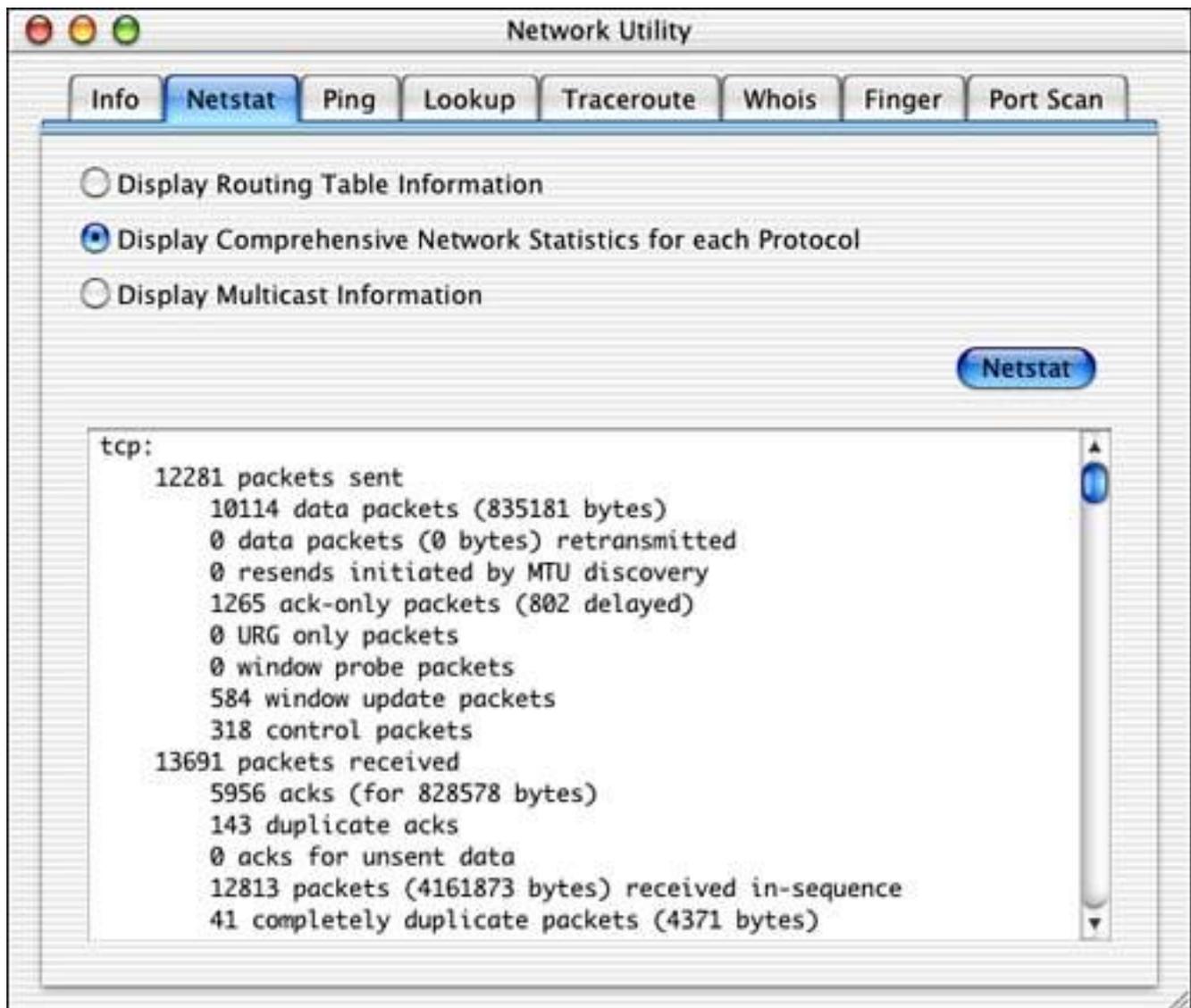
◀ PREVIOUS

NEXT ▶

Industry Standard Protocols

Mac OS X also supports a number of industry-standard network protocols including: TCP, UDP, PPP, PAP, HTTP, FTP, DNS, SLP, DHCP, BOOTP, LDAP, NTP, SOAP, and XML-RPC. The implementation of these protocols means that there are few, if any, limits to Mac OS X connectivity via the Internet. If you don't see it in Mac OS X, the odds are that it can be downloaded, built, and installed in just a few steps. [Chapters 14](#), "XML-RPC," and [15](#), "SOAP," deal with network-based projects that pull data from a remote network server using HTTP, XML-RPC, and SOAP.

Figure 1.7. Network Utility helps keep things running smoothly.



With the plethora of preinstalled software with Mac OS X, you can quickly be running the Apache Web server with full secure connections and Perl or PHP CGI support. An FTP server is just a click away as well. The most common servers are very simple to get up and running and monitor, so no matter what your skill level, you can have a working Internet presence using only Mac OS X. This is a boon for Web-based software developers who need to be able to test their code in a real UNIX environment. Running Apache on your Macintosh desktop is a dream come true for Web developers.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

OpenGL

Mac OS X contains the industry-standard, cross-platform OpenGL library for 3D graphics. OpenGL is a set of high-performance drawing functions optimized for games, medical imaging, CAD/CAM, animation, and other high-end needs. [Figure 1.9](#) shows iTunes drawing colorful animation to a frantic piece by Dizzy Gillespie.

Figure 1.9. Apple iTunes using OpenGL.



Need to write a game with full support for textures and 3D graphics? How about a screen saver that waves a colorful flag in the computerized wind? 3D images of the human body that rotate and allow zooming in, out and between individual cells? OpenGL allows all of this and more. The Mac OS X development tools come with a lot of cool OpenGL examples that are sure to make you want more. Sure, it'll make the fan in your PowerBook come on, but what do you expect?

QuickTime

QuickTime is Apple's original multimedia software allowing the recording, playback, and editing of digital movies, sound, animation, live streaming audio and video, and more. QuickTime makes use of a plug-in architecture to allow it to be expanded and enhanced as necessary to include support for new hardware and software. QuickTime supports a variety of platforms including all newer Macintosh and Windows operating systems. [Figure 1.10](#) shows interactive QuickTime in action.

Figure 1.10. Apple QuickTime Player.



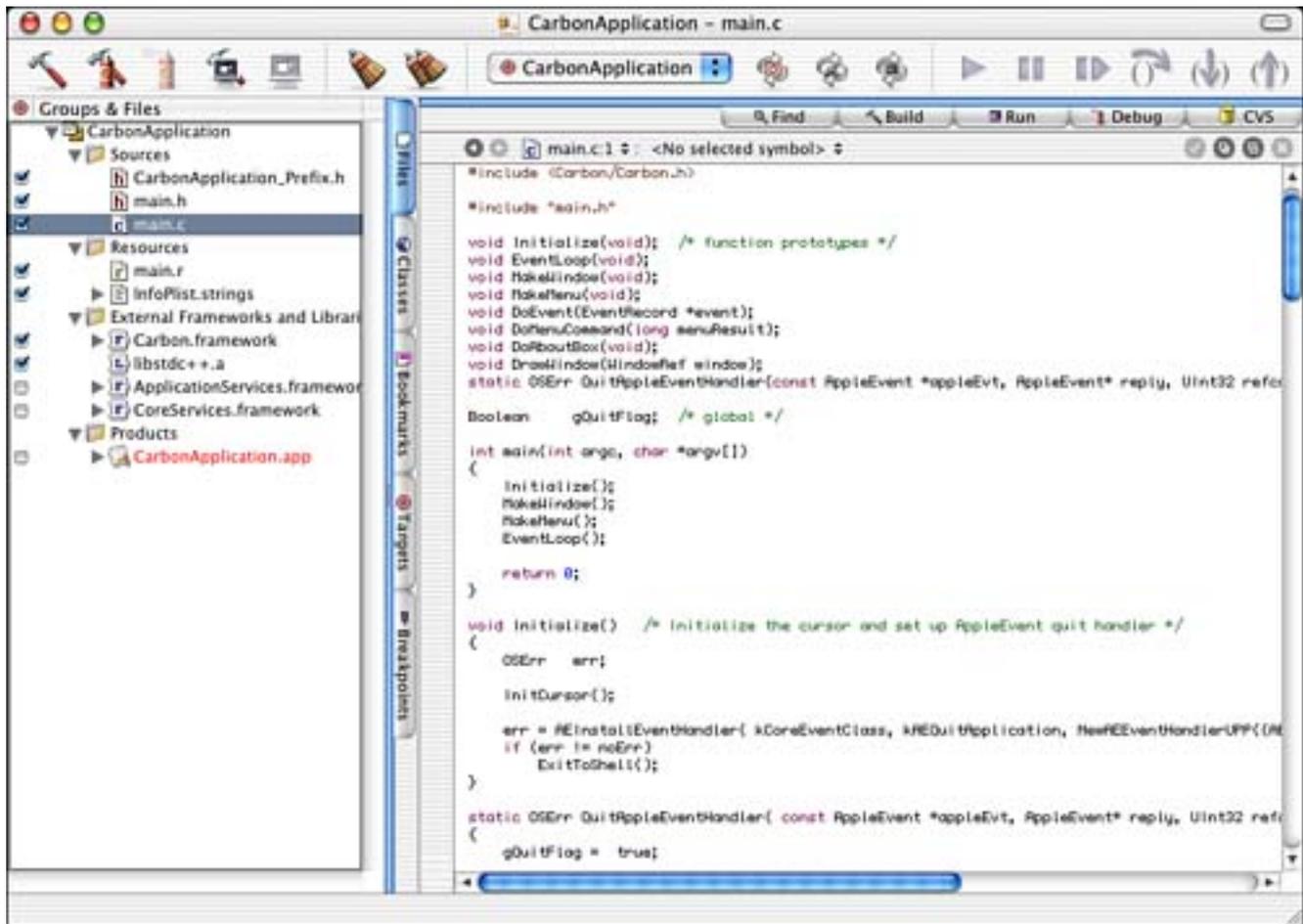
Whether you want to sit back and watch movie trailers, listen to music, or watch a streaming webcast, you can do it with QuickTime. For that matter, using the free iMovie software, you can even create your own QuickTime movies with music, sound, special effects, and titles. Once complete, it only takes a few minutes to upload your creation to a Web site and make it available to the world.

From a developers point of view, you can use the QuickTime frameworks to implement movie playback, editing, and recording from within your application. Your application can support digital video without ever having to write a driver for any video camera—by supporting QuickTime, you have access to everything else that supports QuickTime. QuickTime can also be a great way to implement simple animation in your application or game. The possibilities are endless.

Carbon

The Carbon APIs are based on pre-Mac OS X versions of the system software. These APIs allow developers to take advantage of many Mac OS X features, especially protected memory and preemptive multitasking, while still maintaining support for Mac OS 8.1 and later. The Carbon APIs bridge the gap for developers who have legacy software that they need to move toward Mac OS X. [Figure 1.12](#) shows a Carbon application being developed in Project Builder, one of Apple's free developer tools. [Chapters 5](#), "Carbon Plug-ins," and [6](#), "Frameworks," deal with Carbon-based projects.

Figure 1.12. Creating a Carbon application in Project Builder.



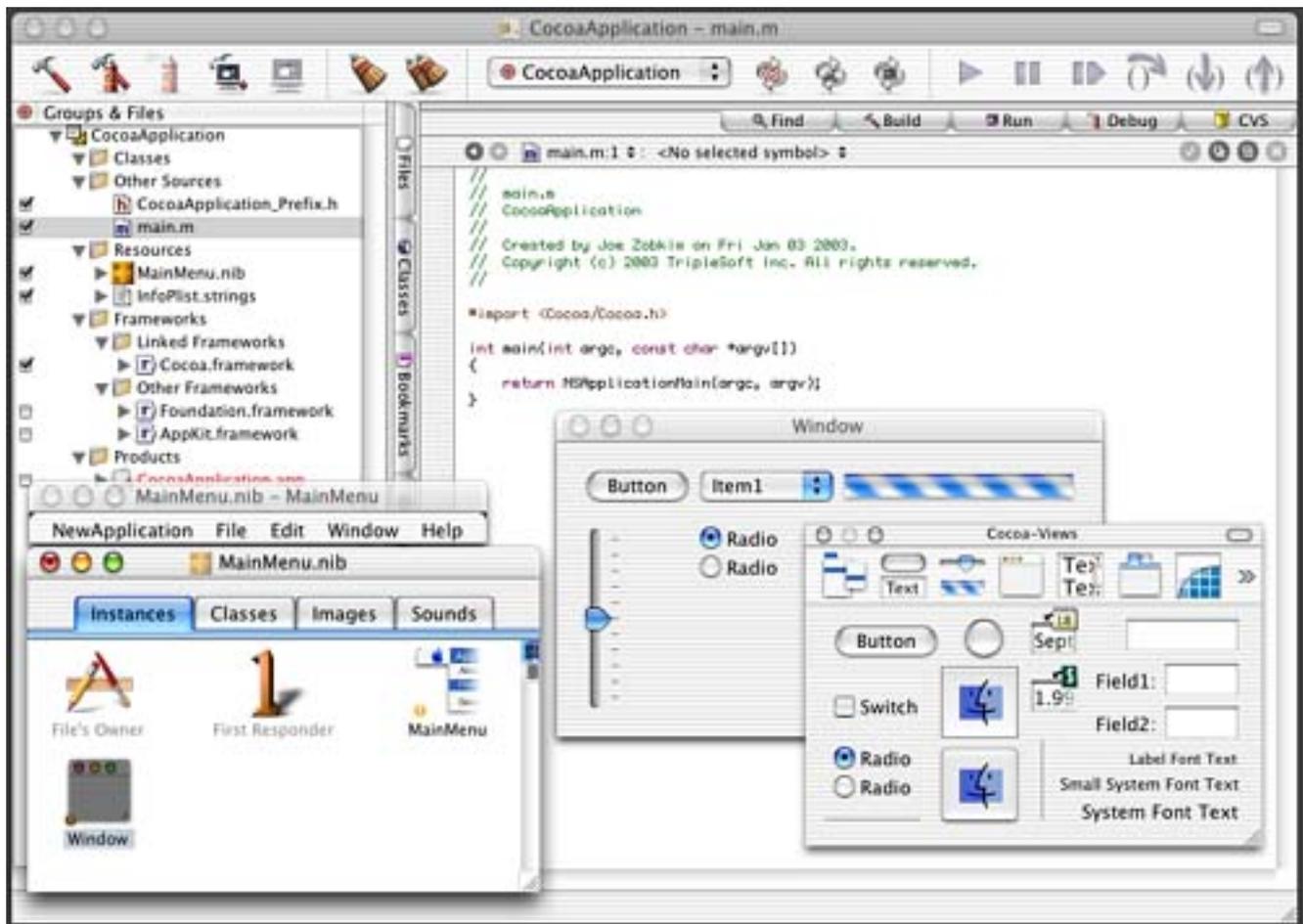
As a developer, you might be responsible for some Macintosh software that you wrote "back in the day" that simply is not ready for Mac OS X. By using the Carbon APIs, you can "Carbonize" your application so that it will still run under these previous OS's while also sporting the new Aqua look and feel under OS X and—most importantly—run under OS X as if it belonged! This saves your users from having to launch the Classic environment each time they want to run your application. I say it again: Developers, avoid writing pre-OS X software if you can.

Cocoa

The Cocoa application environment is the native Mac OS X development environment. It allows rapid application development by the use of tools such as Project Builder and Interface Builder. It also includes many frameworks that make it relatively simple to put together an entire application while writing only the code specific to the application. The Cocoa frameworks handle everything from window management to printing to network access.

Cocoa applications can be developed in either Objective-C or Java. All new application development should be done using Cocoa. You can also call the Carbon APIs from within a Cocoa application if necessary, which can prove very helpful. [Figure 1.13](#) shows a Cocoa application being developed in Project Builder and Interface Builder—two of Apple's free developer tools. Many of the chapters in this book deal with Cocoa-based projects.

Figure 1.13. Creating a Cocoa application in Project Builder and Interface Builder.



The classes used throughout Cocoa development are broken up into two kits. The Application Kit (AppKit) framework contains mostly user interface and graphics oriented classes such as windows, buttons, menus, and drawing primitive classes. The Foundation framework contains mostly non-user interface type classes that pertain to data storage and manipulation or provide some sort of service such as scripting, networking, or messaging.

Cocoa is the future. Cocoa is based on the NEXTSTEP development environment and is simply a joy to use. Using Apple's built-in developer tools gives you a rapid application development environment that still gives you enough to sink your teeth into. This is not some poor attempt at a graphically integrated environment. This is a professional set of tools to create the next generation of killer apps! Embrace Cocoa and live a full life.

[\[Team LiB \]](#)

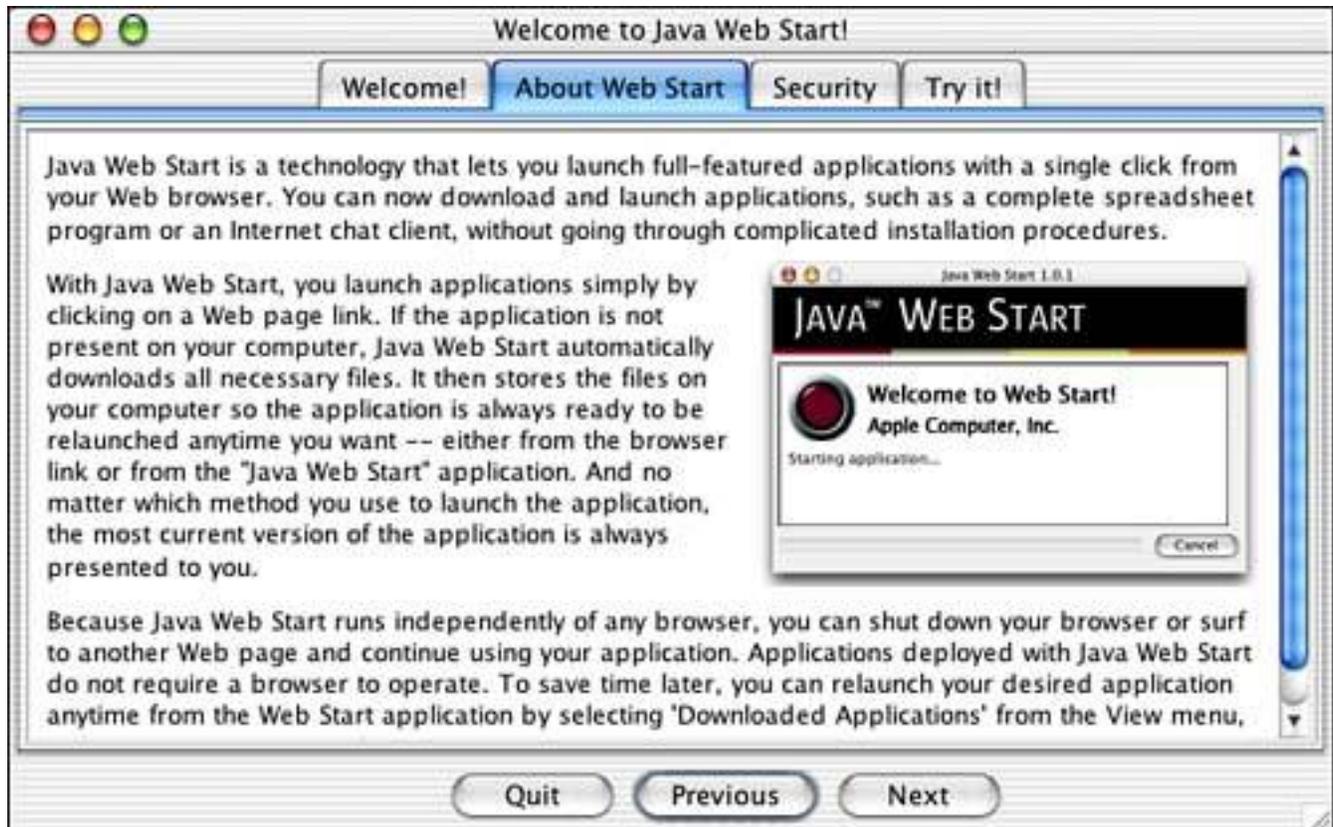
◀ PREVIOUS

NEXT ▶

Java

Mac OS X users can also create 100% Pure Java applications and applets using the Java development environment. Mac OS X supports the Java 2 Standard Edition (J2SE) version 1.4. Write it once in Java, and it will run on a variety of platforms. [Figure 1.14](#) shows the Java Web Start demonstration application that allows you to download and launch full-featured applications from within a Web browser.

Figure 1.14. Java Web Start.



Aqua

Aqua is the name for the Mac OS X user interface. Apple based the interface on the standard elements that Macintosh users expect and added many enhancements to help the novice and advanced user alike. Aqua is the most elegant user interface for any computer yet. [Figure 1.15](#) shows a users home directory open on the Macintosh desktop, a hierarchical menu, and the System Preferences, as well as the dock at the bottom of the screen.

Figure 1.15. Aqua.

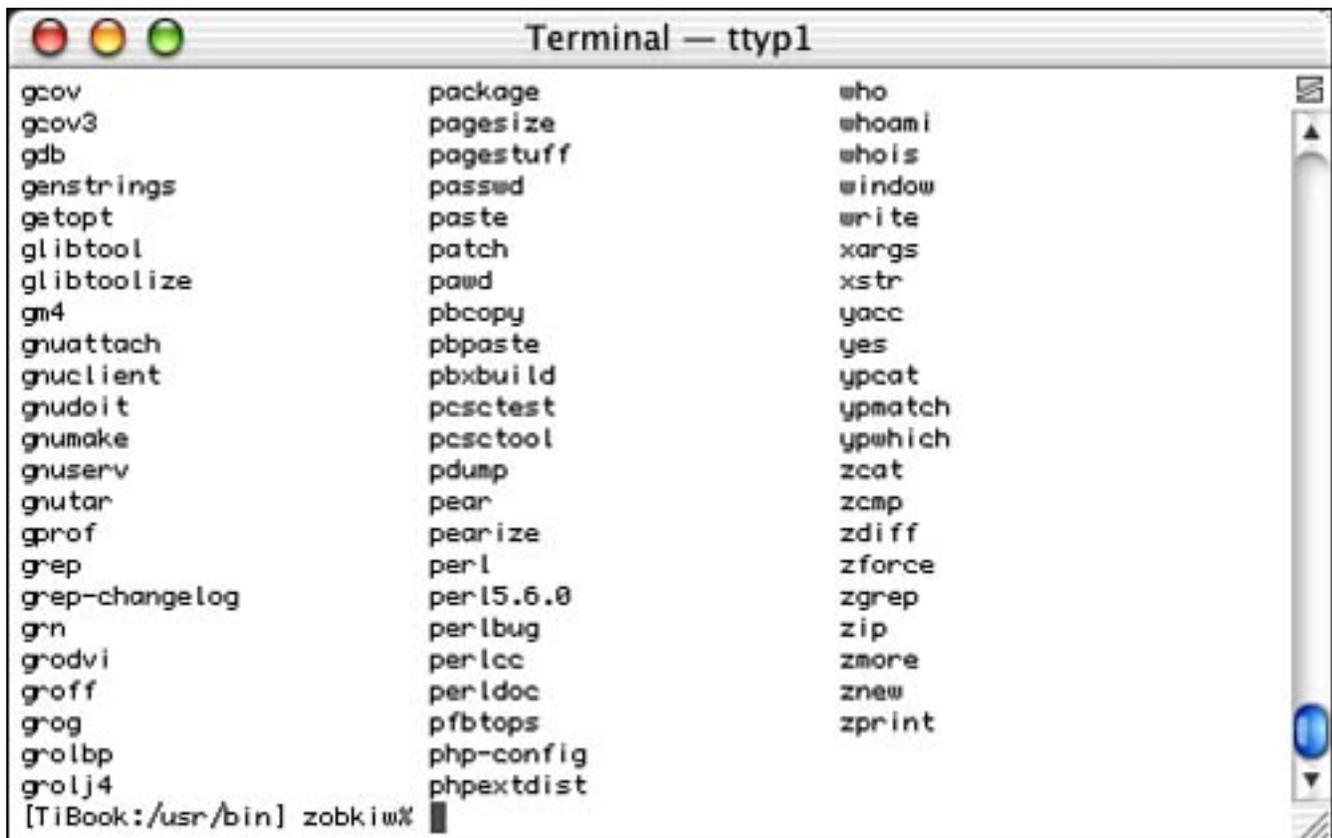


When you first start implementing applications for Mac OS X, it will take some getting used to in regard to following the new Aqua user interface guidelines. Whether you came from the Macintosh, UNIX, NeXT, or Windows worlds, things here are different. Take the time to read Apple's Human Interface Guidelines and practice them to give yourself and your users the best possible experience using your application. Aqua is a fun user interface to work with, and the Interface Builder application makes it a pleasure.

Software

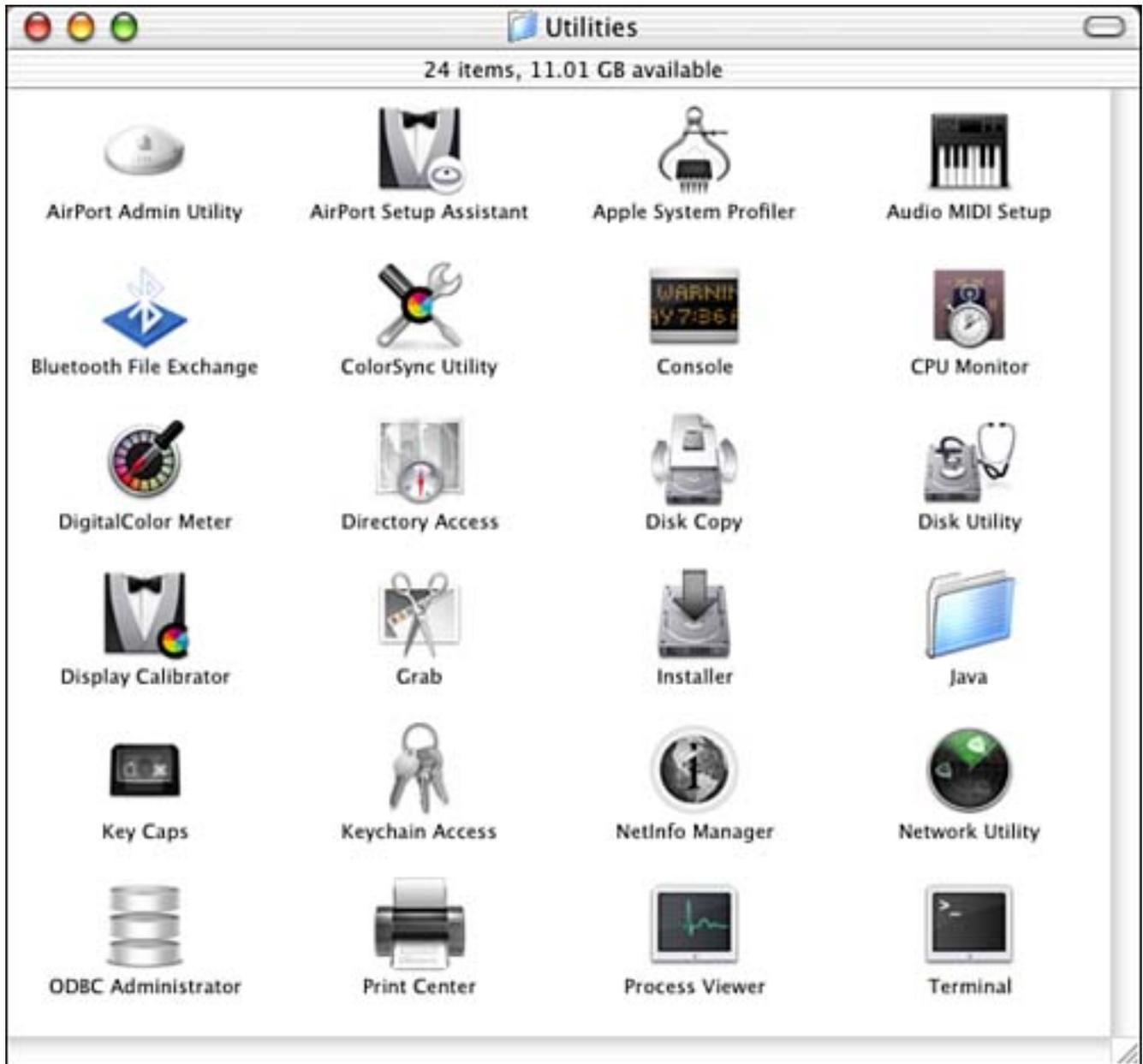
Mac OS X comes with tons (if software can be measured by weight) of industry-standard software preinstalled. Try these on for size: Apache, Perl, PHP, sendmail. These programs and others like them essentially run the Internet. You can now have a complete Web server, email server, and DNS server running on your Macintosh serving files, streaming audio or video and database results to the world. [Figure 1.16](#) shows some of the UNIX applications available on a default Mac OS X installation.

Figure 1.16. About 1/10th of the programs in /usr/bin.



Many utilities, command line and graphical, are included with Mac OS X. UNIX folks as well as Macintosh folks will all feel at home. If you're a GUI person, be sure to dip your toes into the world of the command line. Some things you can only do from the command line. Other things are just more fun to do from the command line. Some GUI applications that come with Mac OS X actually mimic or interface directly to the command line. [Chapter 13](#), "Terminal," shows you how to do this. [Figure 1.17](#) shows some of the GUI applications available on a default Mac OS X installation.

Figure 1.17. Some of the native utilities packaged with Mac OS X.



Conclusion

As you can see, there are many new layers to Mac OS X. Don't plan on learning it all. There was a day when the entire Macintosh operating system fit on a 400K floppy disk. Those were the days when you could know it all. Today, 400K is about the size of some of the icons that represent the software that runs our computers. Don't be overwhelmed; it will come to you as you need it. Just keep exploring and gathering tidbits of information and make use of the excellent Web sites and newsgroups. Macintosh developers are a helpful lot, and you'll find excellent camaraderie with them.

Chapter 2. Introduction to Programming in Mac OS X

"Yeah! Free tools!"

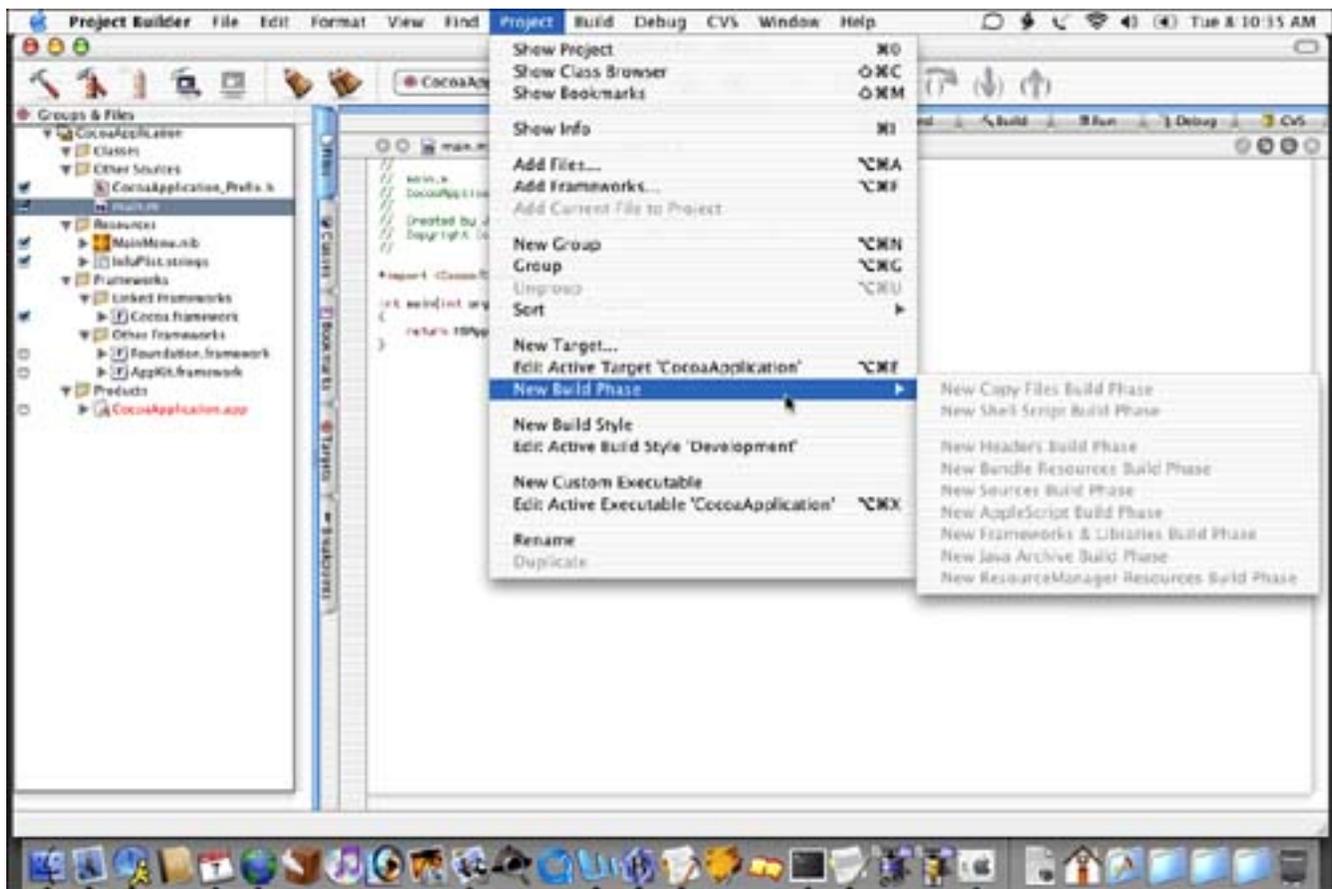
—Anonymous

For those of you coming from the Mac, you might remember MPW (Macintosh Programmers Workshop). This worksheet-style development environment allowed you extremely extensive control over your build configuration. I still remember working on a project that had Assembly, Pascal, and C modules and an incredibly long script that would take hours to build it all on the fastest 68020 of the day. If you ever needed a break, you could "accidentally" touch a well-used header file.

Apple's tools today are much more advanced than MPW, and best of all they're free with Mac OS X. Apple ships a plethora of tools for development, debugging, and exploration. You will mainly use Project Builder and Interface Builder to create and manage your project, but we will touch on many of the available tools in this chapter so that you have a good idea of what is available.

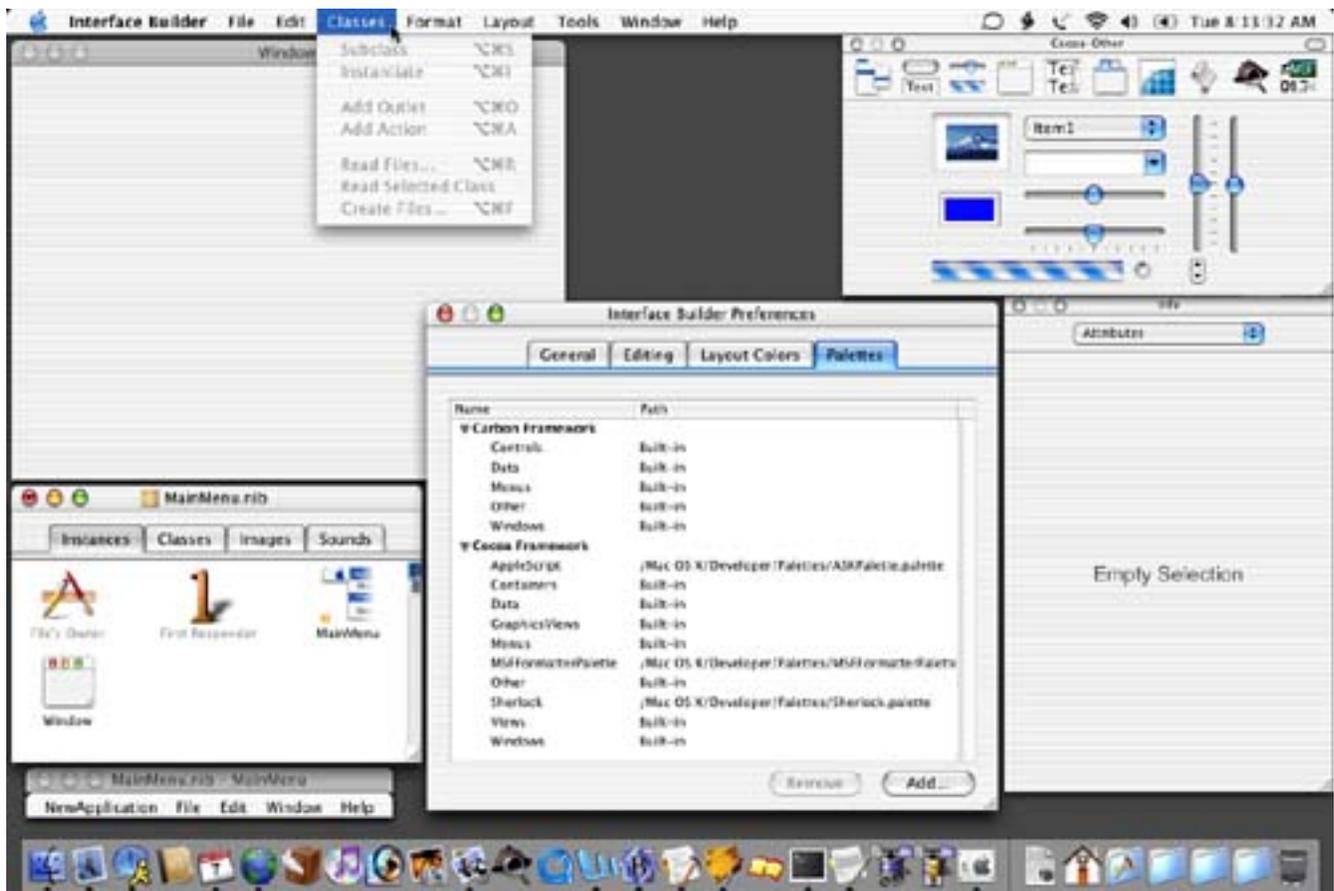
Project Builder (shown in [Figure 2.1](#)) is what you might refer to as an Integrated Development Environment (IDE). It allows you to manage all the source files and resources that go into building your application and create multiple deployment versions of your software. Project Builder allows you to create and manage one-button builds of just about any size project written in a variety of languages including Objective-C, Java, and others. For those of you destined for the command line, you can still use one from within Project Builder features such as debugging. Project Builder uses gcc3 (the GNU Compiler Collection) and gdb (The GNU Debugger).

Figure 2.1. Project Builder.



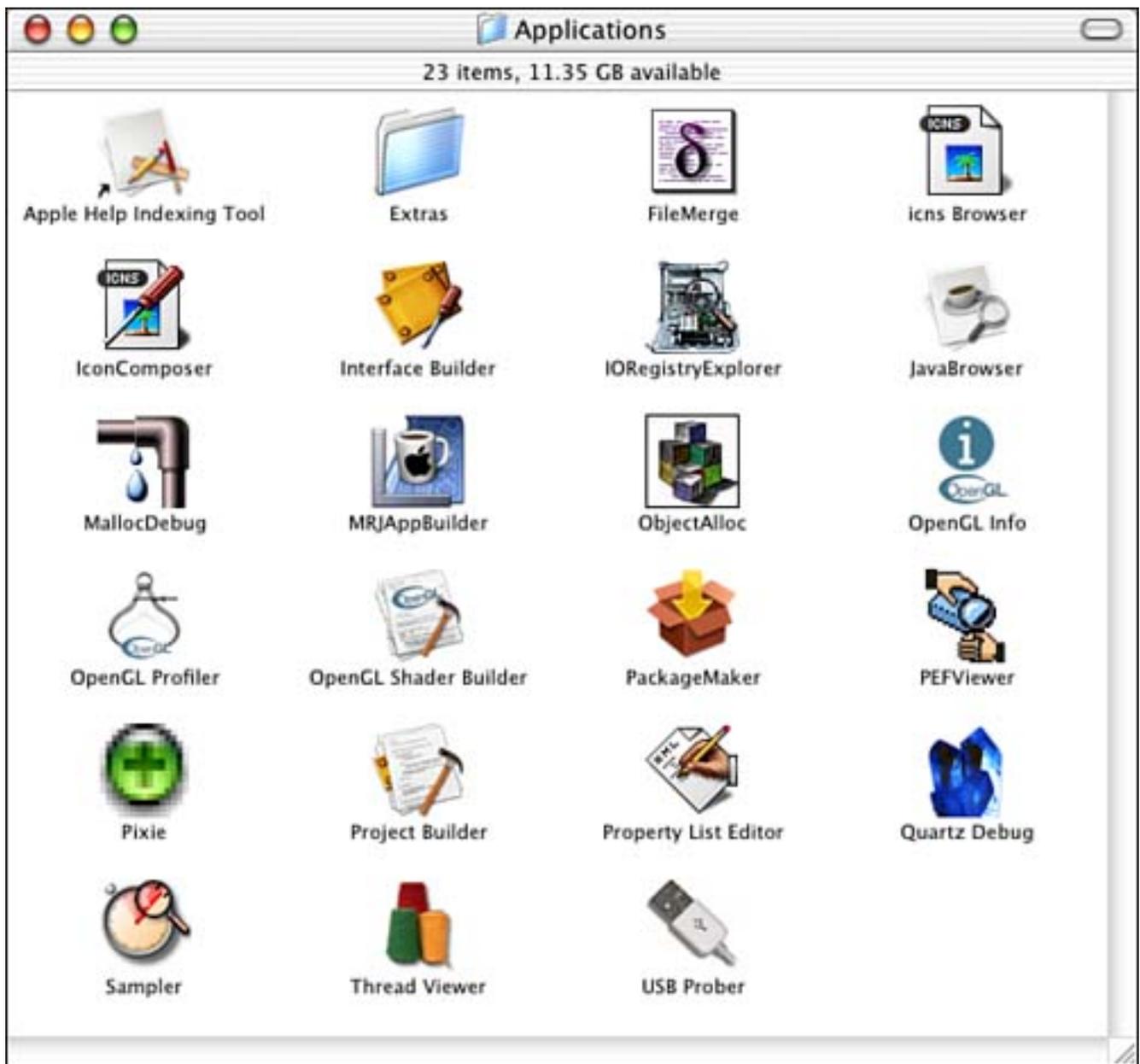
Interface Builder (shown in [Figure 2.2](#)) is the main tool you use to create your application's user interface. Using Interface Builder, you create windows, lay out the controls within them, link your class instance variables (outlets) and methods (actions) to your user interface elements, and more. Interface Builder is also used to edit menu items and set the attributes of even the simplest buttons. It is extensible and supports plug-ins, which allow you to implement your own custom objects. Interface Builder is kind of like a much more sophisticated version of ResEdit.

Figure 2.2. Interface Builder.



[Figure 2.3](#) shows some of the other development tools packaged with the standard development kit. These tools are mostly small utilities used for specialized purposes. Not everyone will use every tool in this collection, but when needed, you'll be thankful that they're there. These utilities will be looked at in more depth later in this chapter.

Figure 2.3. More Development Tools.



Programming Languages

When developing software for Mac OS X, you have a few different development languages to choose.

Objective-C

Objective-C and the Cocoa framework should probably be your first choice for new development projects. Objective-C is not hard to learn if you know C and even easier if you know C++ or some other object-oriented programming language. You can also use straight C and C++ especially if you are maintaining a Carbon-based development project. [Listings 2.1](#) and [2.2](#) show a simple Objective-C class.

Listing 2.1 MyClass.h—an Objective-C Header File

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

@interface MyClass : NSObject {
    BOOL locked;
}

- (void)setLocked: (BOOL)new_locked;
- (BOOL)locked;

@end
```

[Listing 2.1](#) defines the header file of an Objective-C class. After `#import`ing the standard `Foundation.h` header file, we define the `interface` for our class. The class is named `MyClass` and is a subclass of the `NSObject` base class. `MyClass` has one instance variable, a `BOOL` named `locked`. It also has two methods—one to set the value of `locked` and one to retrieve the value of `locked`. Note that Cocoa convention defines how we name these two methods. We end with the `@end`.

Listing 2.2 MyClass.m—an Objective-C Source File

```
#import "MyClass.h"

@implementation MyClass

- (void)setLocked: (BOOL)new_locked
{
    locked = new_locked;
}

- (BOOL)locked
{
    return [self locked];
}

@end
```

[Listing 2.2](#) defines the source file of the aforementioned Objective-C class. We first `#import` the header file `MyClass.h` (see [Listing 2.1](#)) and then define the `implementation` of `MyClass`. This simply involves defining the two methods `-setLocked:` and `-locked`. `-setLocked:` takes the `new_locked` value and sets the instance variable `locked` to that value. `-locked` returns the current value of the `locked` instance variable. Note the name of the "get" method is the same as the instance variable it is getting; this is by Cocoa convention.

If you are new to Objective-C, once you browse some source code, it will all come together quite quickly for you. There are also some great online and offline tutorials on learning the language, including an entire manual in PDF format available from the Apple developer Web site, which can be found at <http://developer.apple.com/>.

Java

Project Builder contains many templates for using Java as your language of choice. Java can be used to develop Cocoa applications or Java applets and applications. Although details are sketchy, not many developers seem to be using Java with Cocoa. However, plenty are using Java to write cross-platform Java applets and applications. The Macintosh can now be considered a reliable platform to develop on using Java, and it improves daily.

[\[Team LiB \]](#)

◀ PREVIOUS NEXT ▶

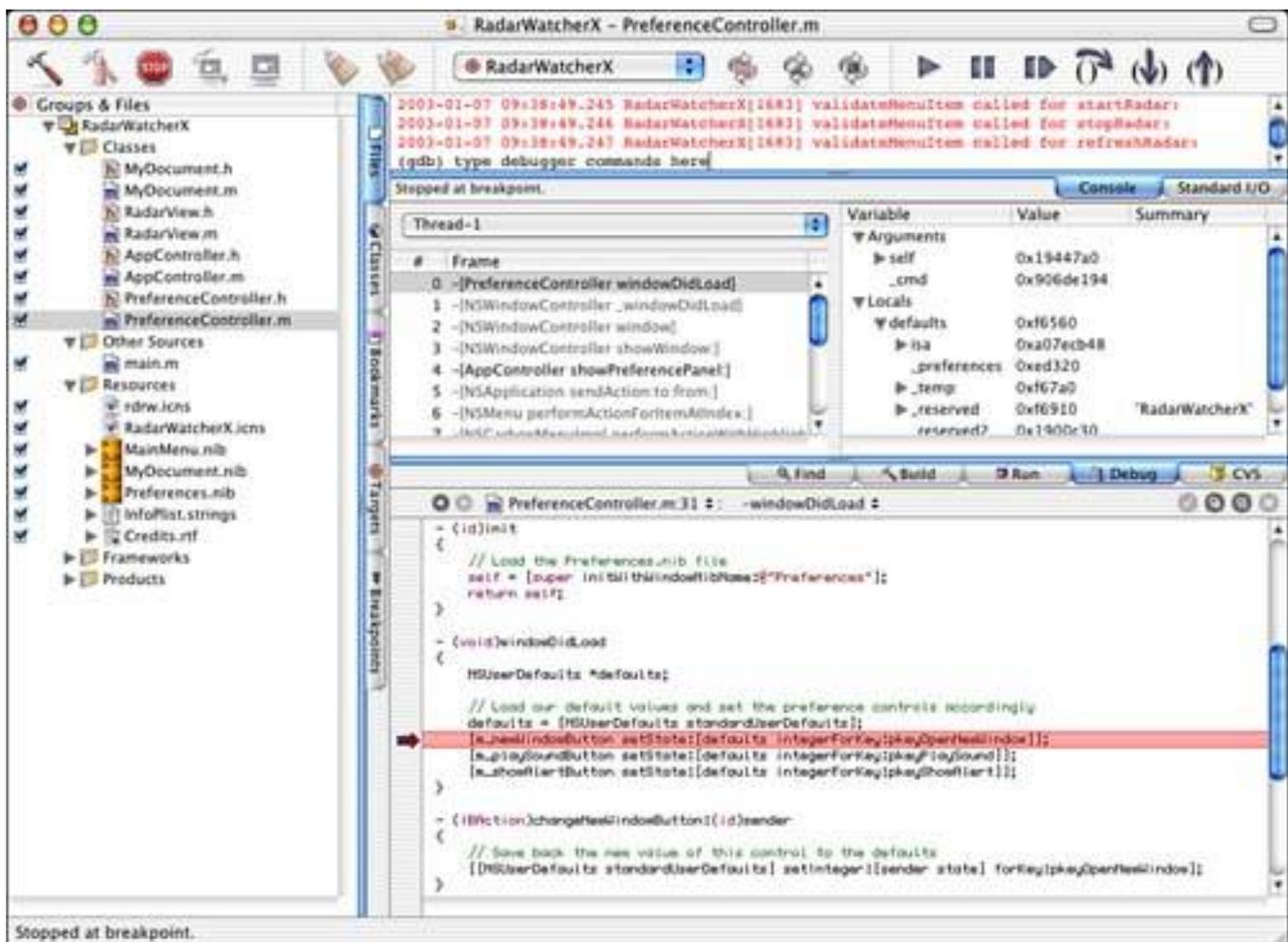
Tools

We've looked briefly at the two main tools available for development under Mac OS X—Project Builder and Interface Builder. Now let's take a look at some of the many other faces in your development arsenal, ranging from tabs within Project Builder to standalone utility applications.

Debugging

[Figure 2.4](#) shows Project Builder in Debug mode. The gdb (GNU Debugger) is the debugger of choice for use in Project Builder. Through the Debug tab, you can view the current stack frame and threads in addition to a hierarchical list of global and local variables and method arguments as well as their values.

Figure 2.4. Debugging in Project Builder.



By using the Console tab, you can type commands directly into gdb as well as view debugging statements from your program. Although Project Builder allows you to do many of the standard debugging practices from the graphical user interface, such as setting breakpoints, typing commands directly into gdb allows you to completely master the environment your program is running in. The gdb is an incredibly powerful tool that is worth your time to learn.

Type `man gdb` in Terminal for more information.

Source Control

Project Builder contains built-in support for CVS, the Concurrent Versions System. CVS is a version control system that allows developers to keep a log of who changed what file as well as when and how. CVS supports hierarchical collections of directories and files and is especially useful when managing a project among multiple authors. CVS works over a local or wide area network including the Internet.

In Mac OS X, you can use CVS from the command line as well as directly from within Project Builder. CVS is very much a command line tool, especially when setting up a new repository. There are detailed instructions packaged with the development tools that walk you through setting up a new project using CVS.

Type `man cvs` in Terminal for more information.

Developer Utilities

In [Figure 2.3](#), you saw a collection of developer utilities that are made available by Apple. Let's look a bit more closely at each of these tools.

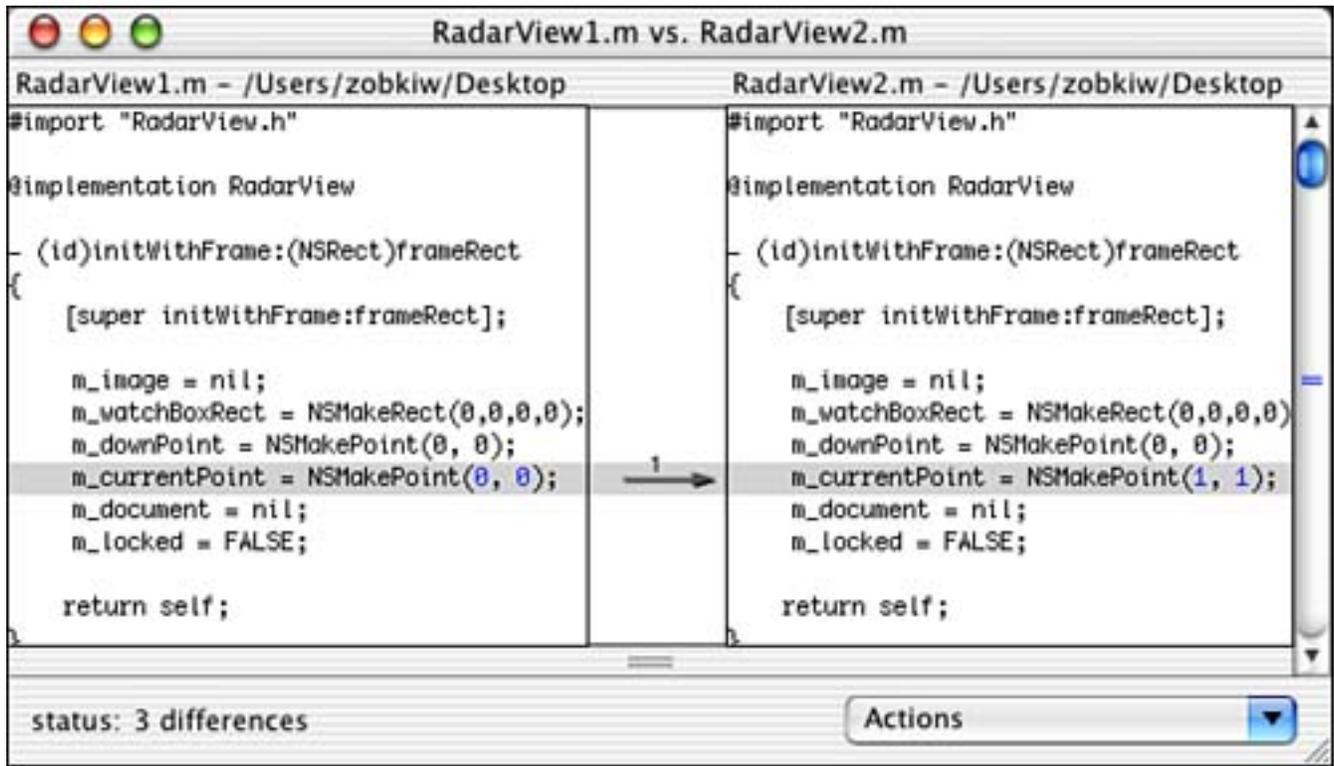
The **Apple Help Indexing Tool** is used to create a search index for Help Viewer files. Help Viewer can be seen by selecting Mac Help from the Finder's Help menu. Under Mac OS X, this is the standard way to provide online, searchable, linked help (see [Figure 2.5](#)).

Figure 2.5. The Help Viewer main window.



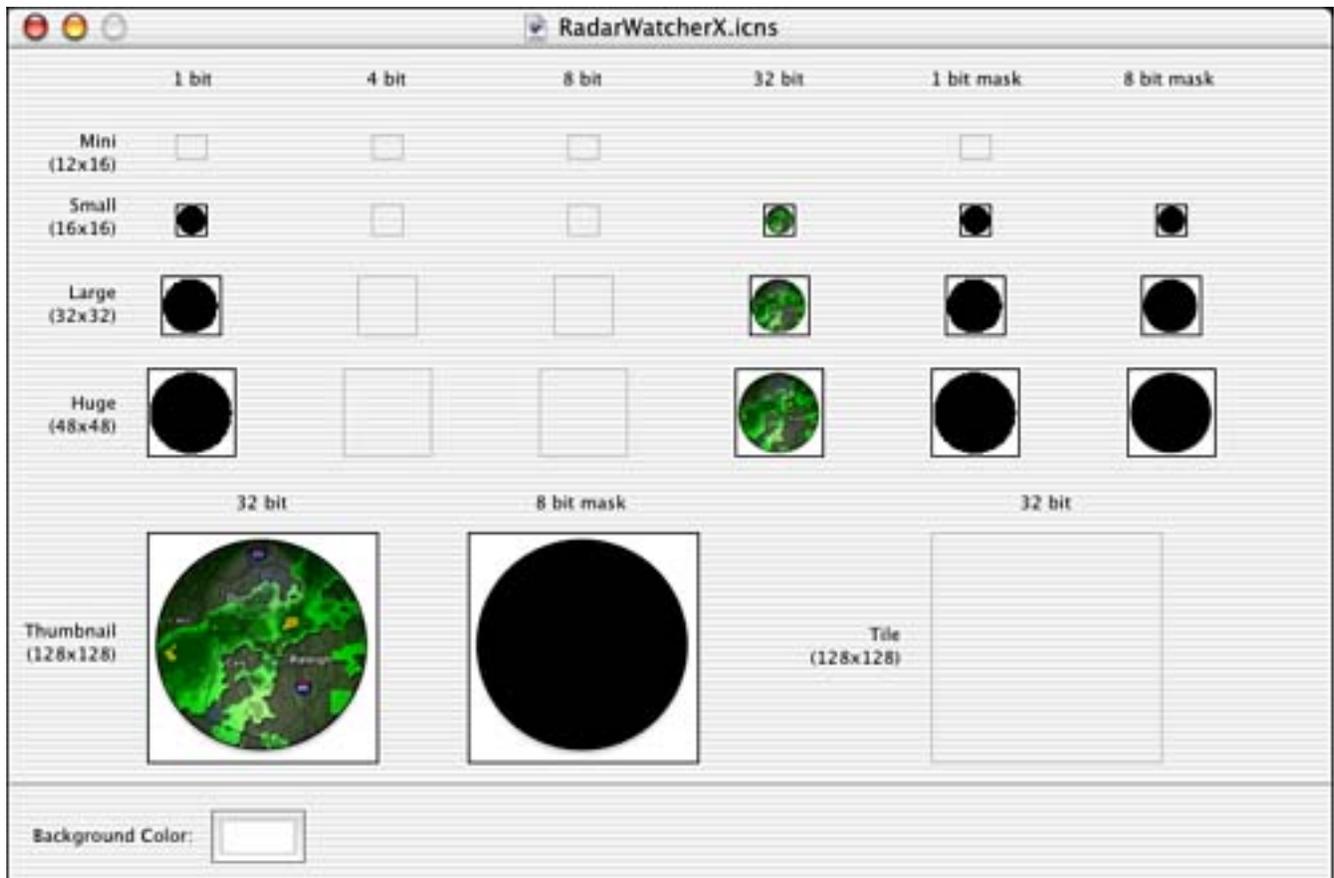
FileMerge allows you to easily compare two files for differences. After selecting the two files to compare, FileMerge will quickly locate the differences between the files and allow you to merge the changes into one. For each difference, you can choose either the old or the new version. This can be very handy when you receive a file back from a developer and you're not sure what he changed (or what you changed since the file was out of your control). [Figure 2.6](#) shows a comparison in progress.

Figure 2.6. The FileMerge differences window.



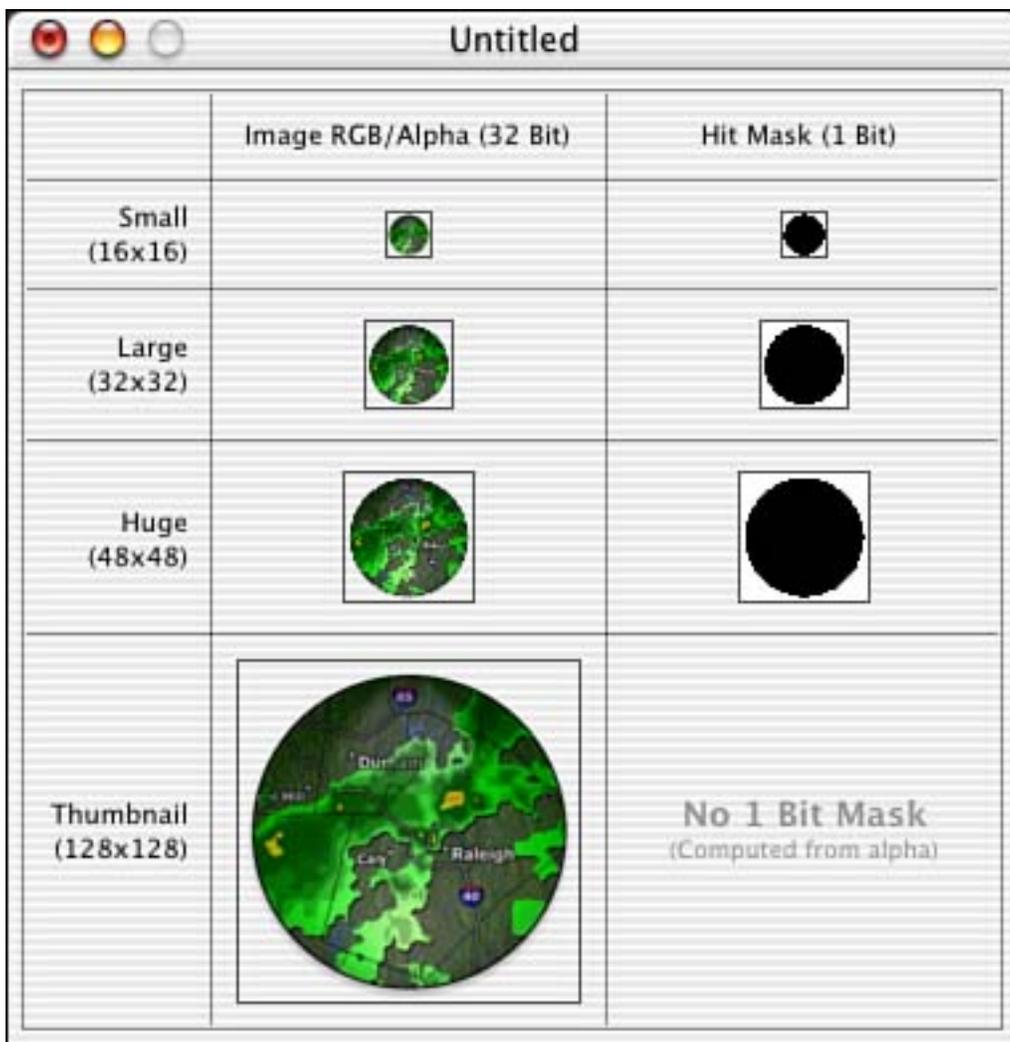
Icns Browser allows you to view the contents of any icns file. These files normally contain application or document icons in various sizes and bit depths. [Figure 2.7](#) shows the icns Browser in action.

Figure 2.7. The icns Browser window.



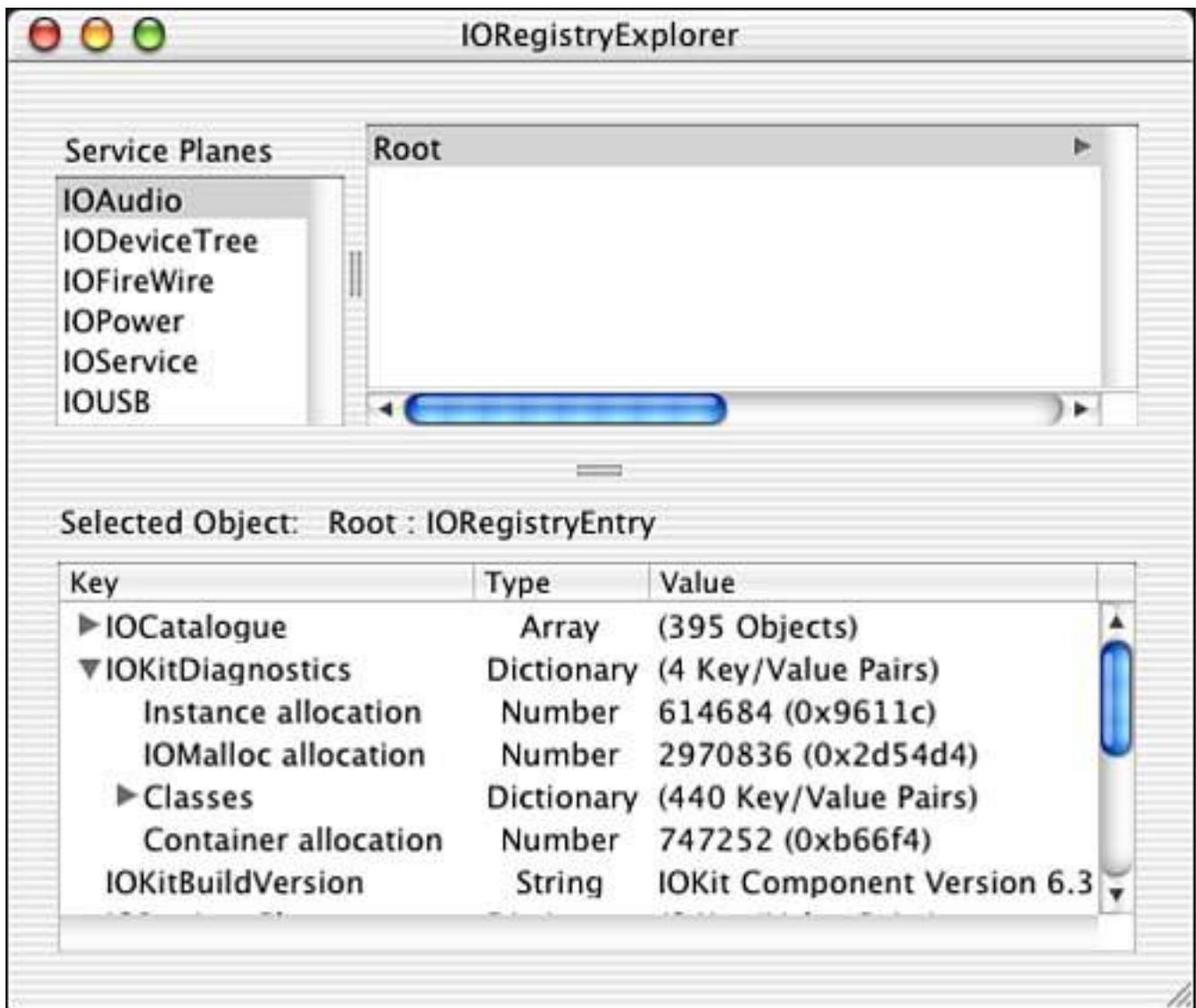
IconComposer creates the icns files that we browse with icns Browser. By importing standard formats such as TIFF or JPG, you piece together the various sizes and bit depths to create the finished icns file (see [Figure 2.8](#)).

Figure 2.8. The IconComposer window.



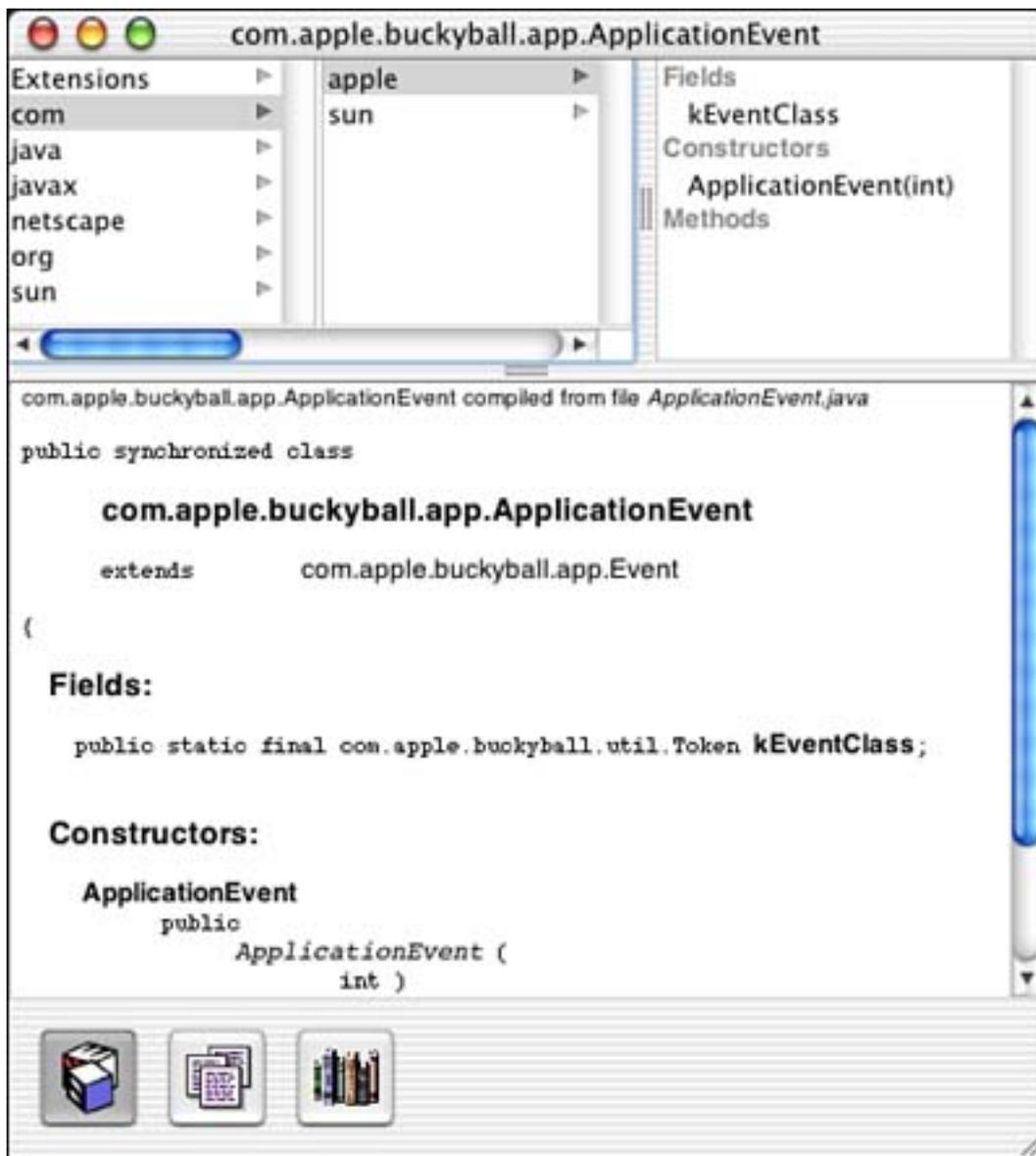
IORegistryExplorer is a tool for use by developers writing I/O drivers. This tool allows you to browse, search, and update the I/O registry (see [Figure 2.9](#)).

Figure 2.9. The IORegistryExplorer window.



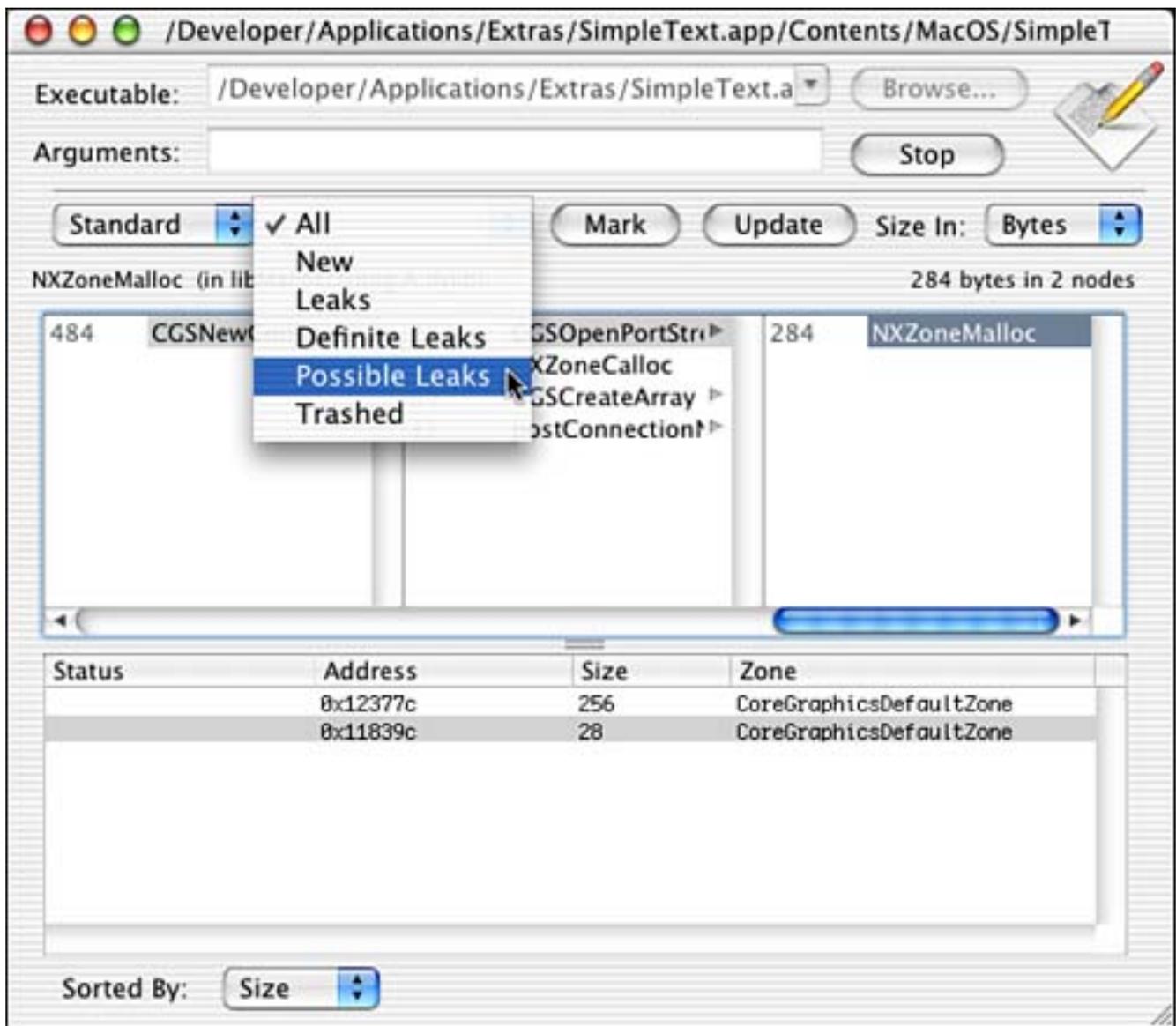
JavaBrowser allows Java developers to browse Java classes, documentation, and source files. [Figure 2.10](#) shows an application source file being browsed.

Figure 2.10. The JavaBrowser window.



MallocDebug is a utility to help in the understanding of how an application uses memory. It can measure and analyze all allocated memory in an application or measure the memory allocated since a given point in time. It can also be used to detect memory leaks. [Figure 2.11](#) shows MallocDebug in the process of analyzing an application.

Figure 2.11. The MallocDebug window.



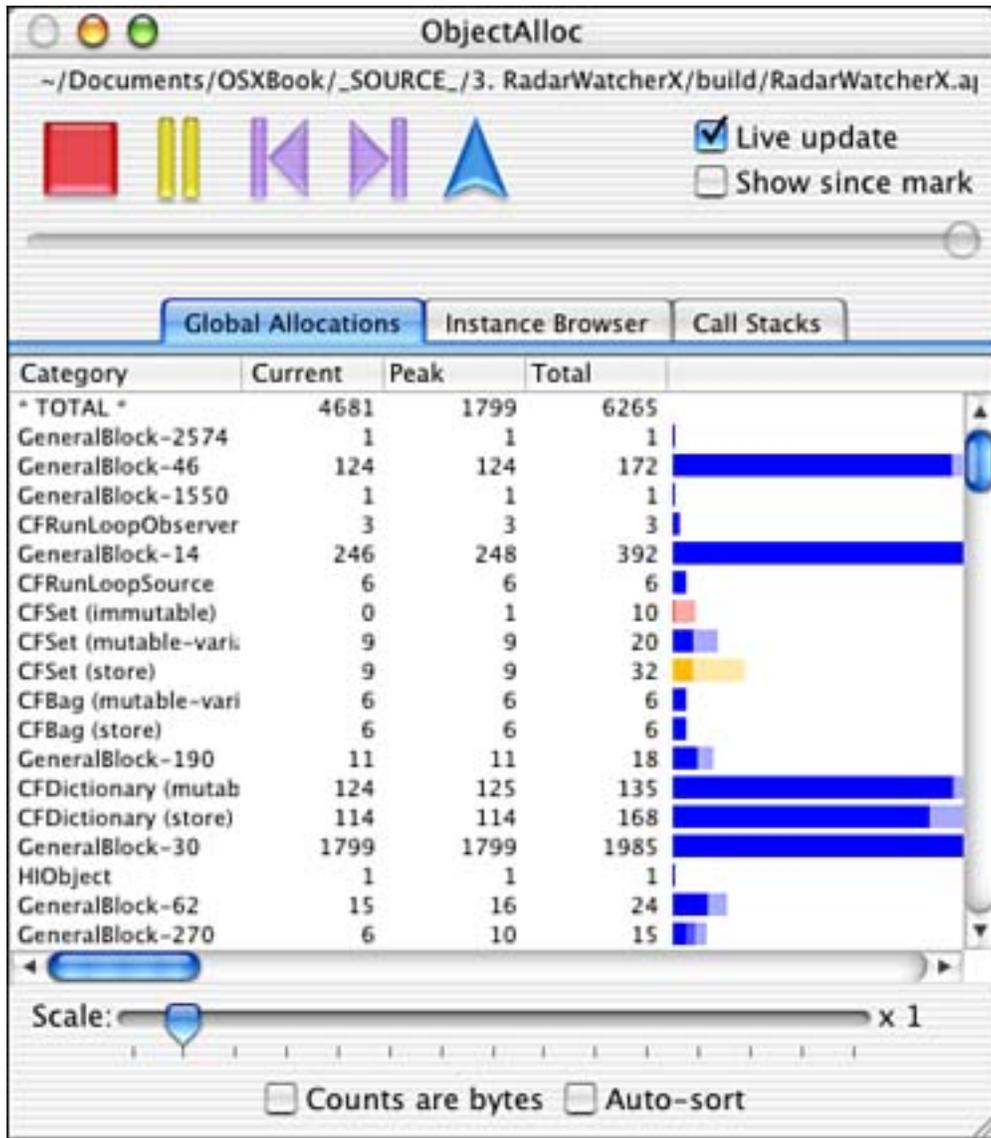
JarBundler is new with Java 1.4 for Mac OS X and is used by Java developers to package their program's files and resources into single double-clickable applications. By using Mac OS X's bundle technology, developers can make their Java applications simpler to install and use with this utility (see [Figure 2.12](#)).

Figure 2.12. The Jar Bundler window.



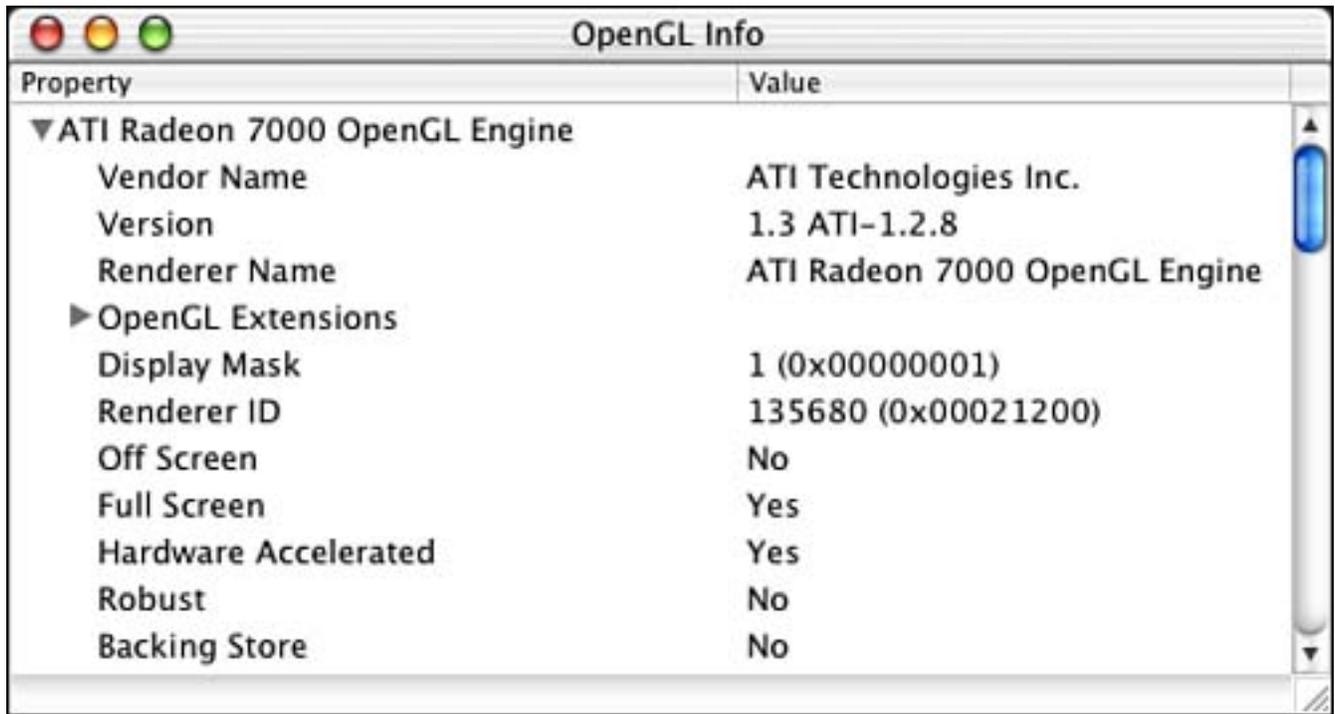
ObjectAlloc allows developers to observe memory allocations and deallocations in an application in almost real-time. You can also view the history of memory use over time to help identify allocation patterns throughout the life of an application. [Figure 2.13](#) shows ObjectAlloc in the process of analyzing an application being launched.

Figure 2.13. The ObjectAlloc window.



OpenGL Info displays information about the installed OpenGL extensions and is useful to developers working with OpenGL (see [Figure 2.14](#)).

Figure 2.14. The OpenGL Info window.

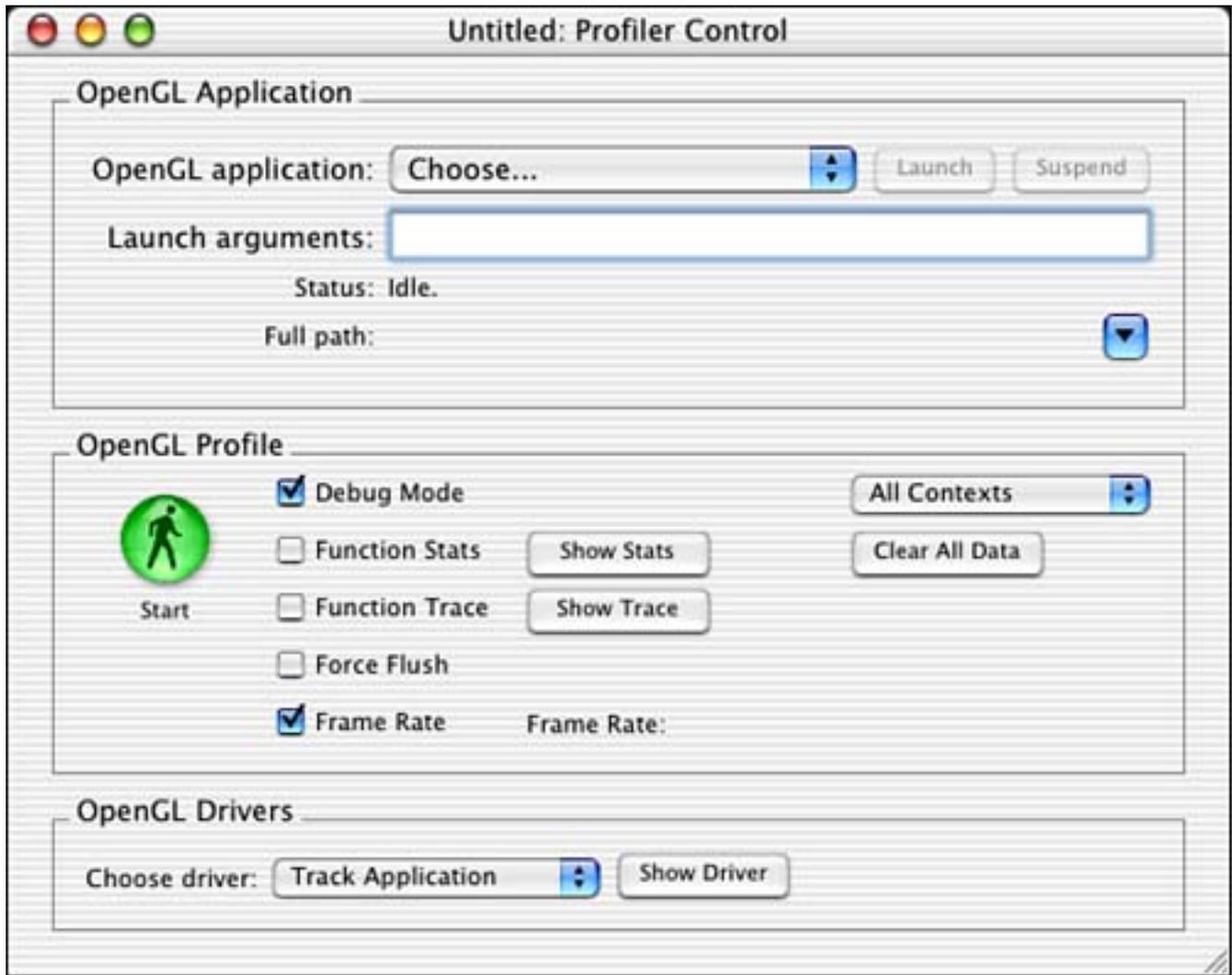


The image shows a window titled "OpenGL Info" with a table of properties and values. The table has two columns: "Property" and "Value". The properties listed are:

Property	Value
▼ ATI Radeon 7000 OpenGL Engine	
Vendor Name	ATI Technologies Inc.
Version	1.3 ATI-1.2.8
Renderer Name	ATI Radeon 7000 OpenGL Engine
► OpenGL Extensions	
Display Mask	1 (0x00000001)
Renderer ID	135680 (0x00021200)
Off Screen	No
Full Screen	Yes
Hardware Accelerated	Yes
Robust	No
Backing Store	No

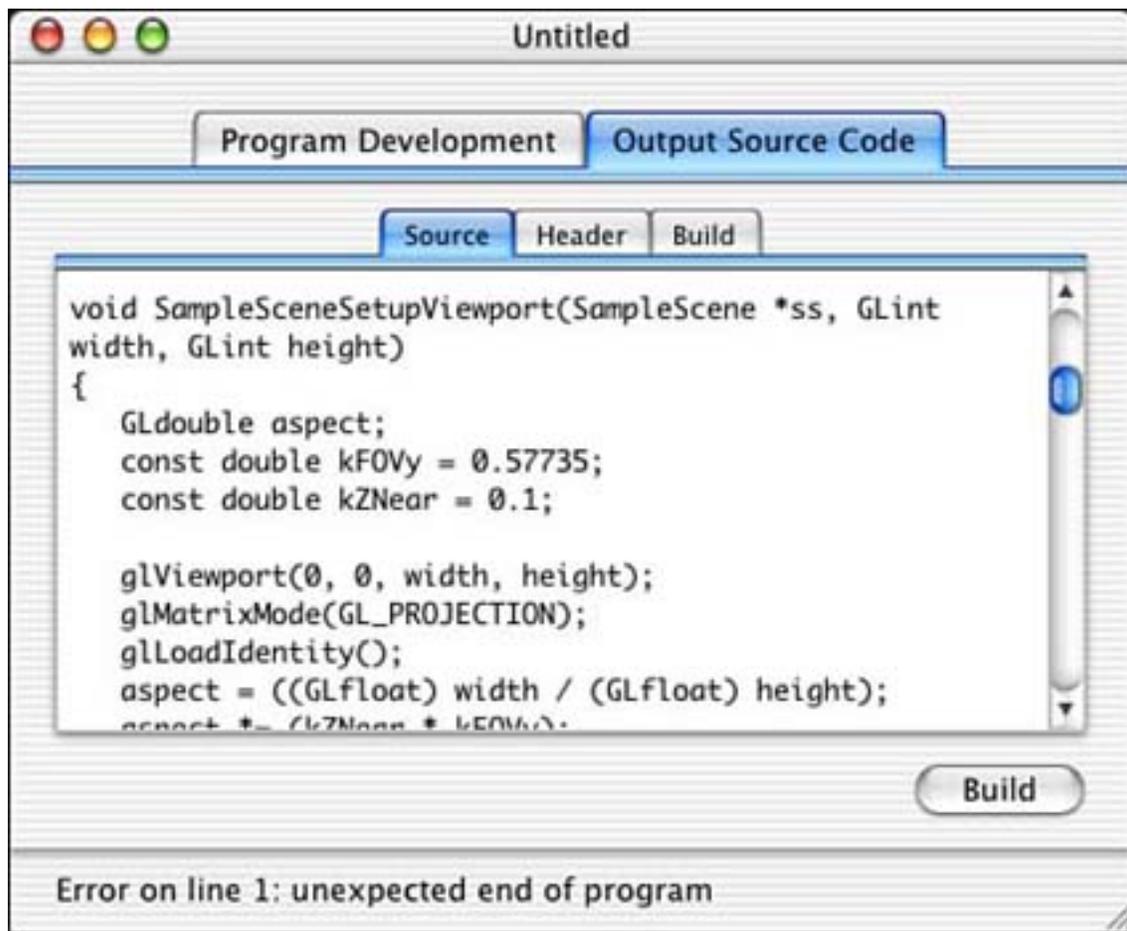
OpenGL Profiler allows you to profile an application that is using OpenGL. Because OpenGL can be so processor intensive, the more you can optimize an OpenGL application, the better. This utility is just the right one for the job. [Figure 2.15](#) shows the Profiler Control window, but the application also provides charts and graphs of application activity once the profile begins.

Figure 2.15. The OpenGL Profiler window.



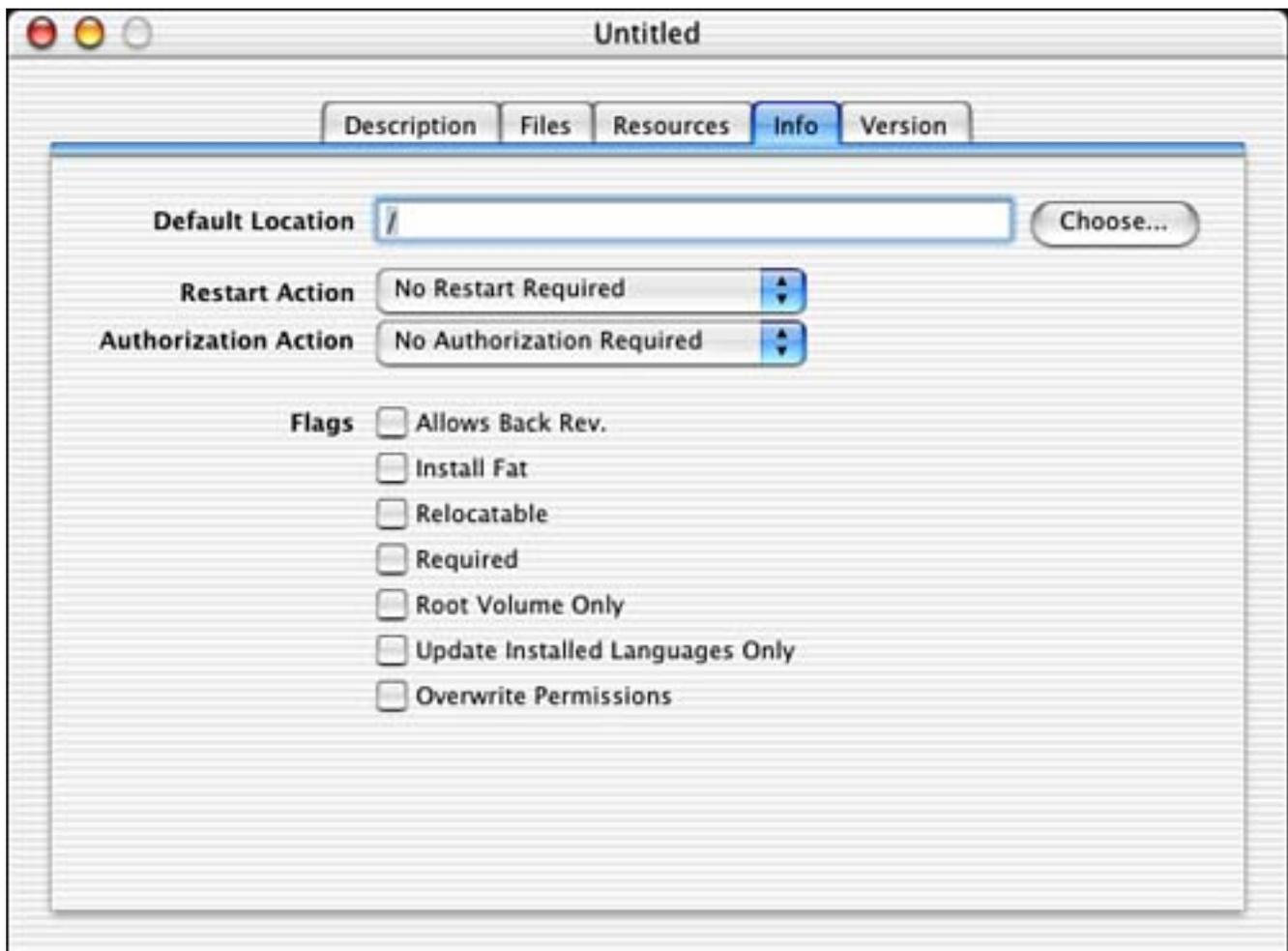
OpenGL Shader Builder allows real-time entry, syntax checking, debugging, and analysis of vertex/fragment programs for use with OpenGL. You can also export your creation to a sample GLUT application in Project Builder format. It also includes an instruction reference (see [Figure 2.16](#)).

Figure 2.16. The OpenGL Shader Builder window.



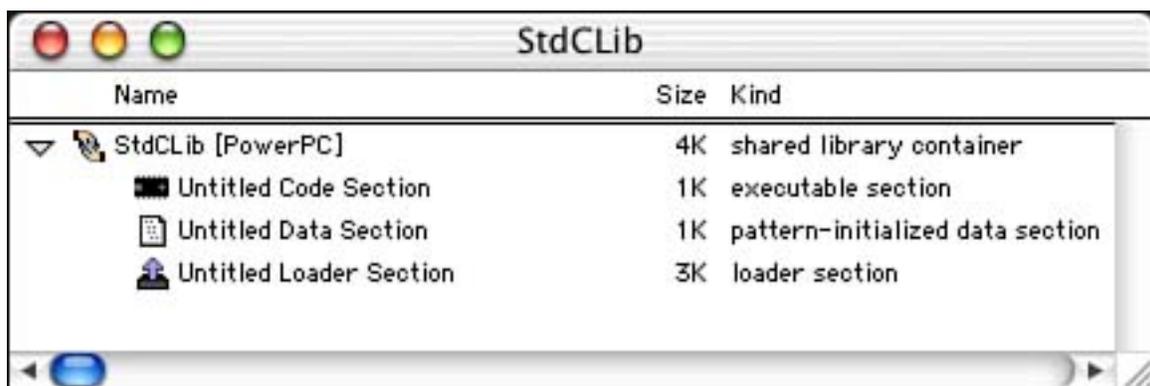
PackageMaker, brought to you by the Apple Mac OS X Installer Elves, allows you to easily create installers for your applications. [Figure 2.17](#) shows the Info tab of the PackageMaker application. Note that you can require a restart if needed or authorization before allowing a package to be installed. Plenty of other options are also available, so make sure that you read the documentation on using this powerful tool.

Figure 2.17. The PackageMaker window.



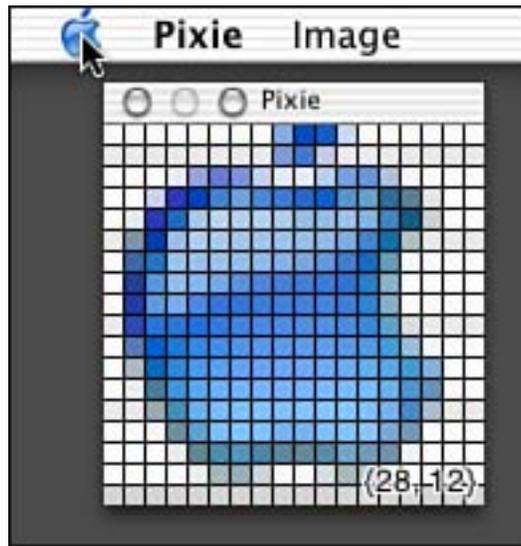
PEFViewer allows you to easily browse and search detailed information and HEX views of any Preferred Executable Format file. [Figure 2.18](#) shows PEFViewer looking inside the small StdCLib file.

Figure 2.18. The PEFViewer window.



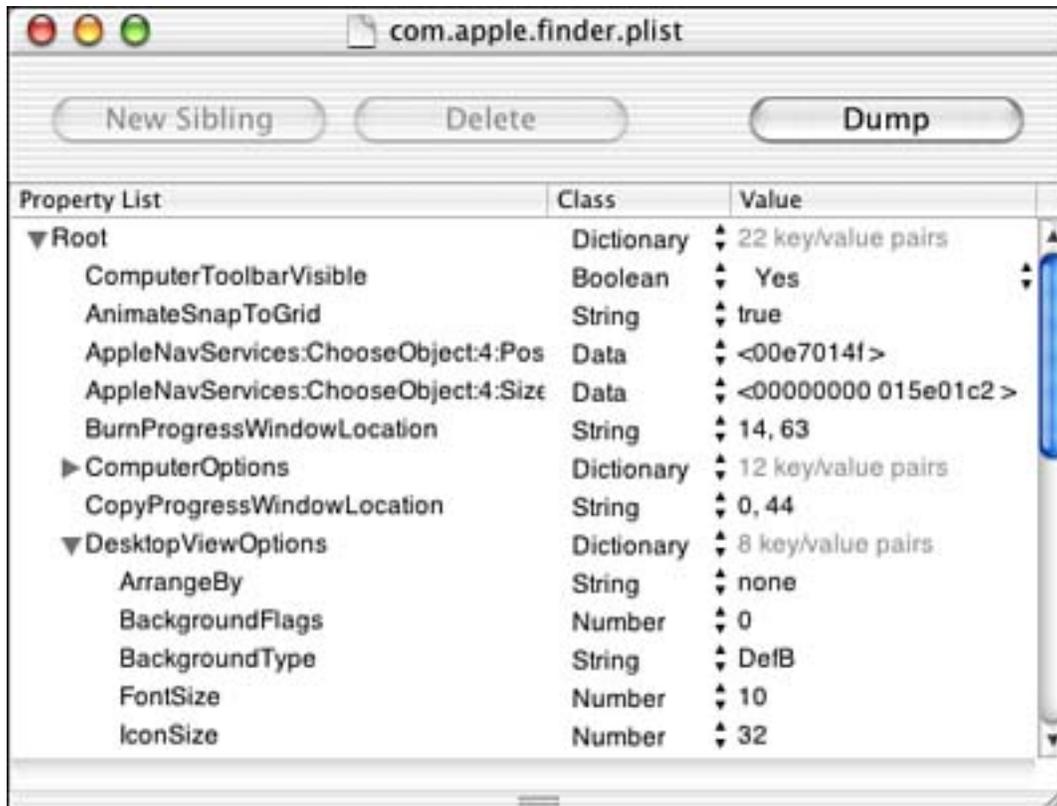
Pixie is a great little application that allows you to zoom in at various levels to view the pixels underneath the cursor. You can see the information in a variety of ways and export the selected area to a TIFF file. This tool is great for examining your user interface in minute detail. [Figure 2.19](#) shows Pixie in action under the Apple icon.

Figure 2.19. The Pixie window.



Property List Editor allows you to view property list files in the true hierarchical fashion that they were meant to be viewed. Property list files are nothing more than XML-format text files. However, if the thought of viewing them in a text editor sends shivers up your spine, the Property List Editor is here to take some of the shiver away. You still need to know what you're doing, but it will take you a few seconds more to completely screw up your machine using this handy little utility as opposed to a text editor (see [Figure 2.20](#)).

Figure 2.20. The Property List Editor window.



Quartz Debug is a great utility that helps you debug the Quartz drawing in your application. This includes window refreshes and just about any other 2D drawing you do in your program. By enabling the Flash screen updates or Flash identical updates options, you can see exactly where the screen is being refreshed

throughout the operating system. [Figure 2.21](#) shows the Quartz Debug palette, but there is also a view that displays detailed information about each program's open windows.

Figure 2.21. The Quartz Debug window.

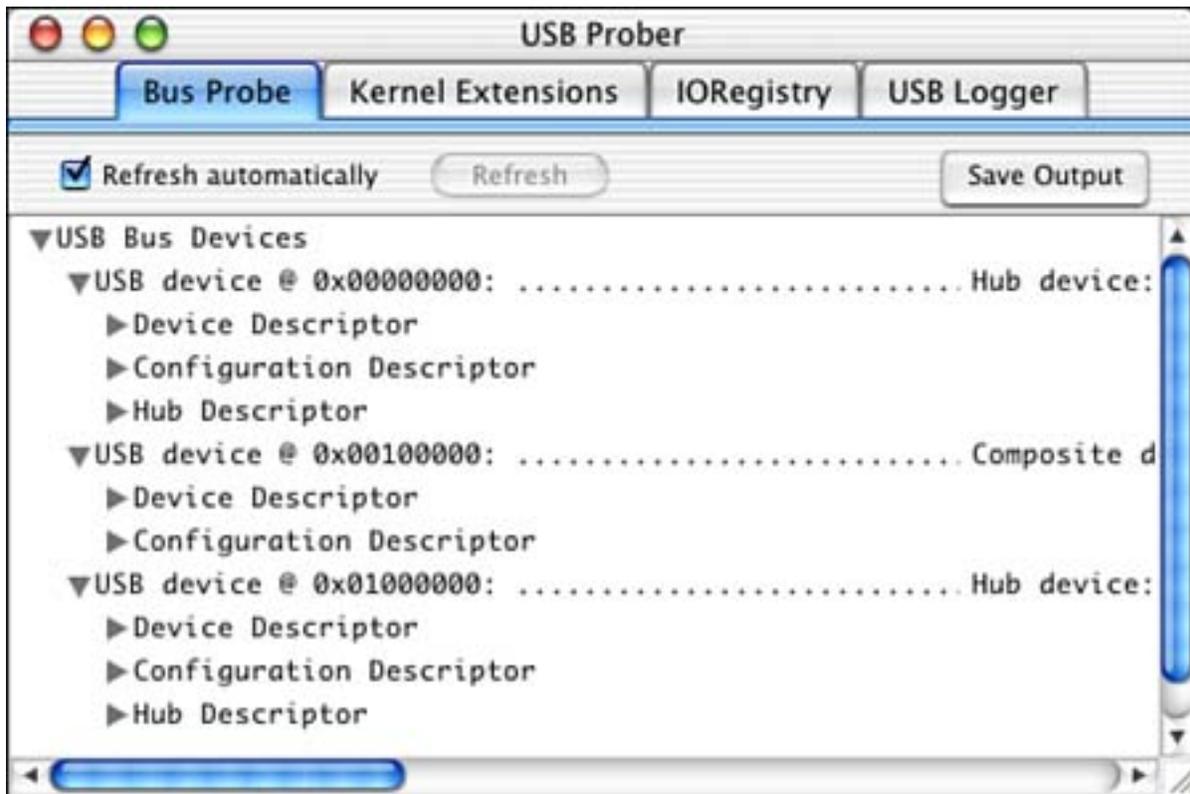


Sampler is a utility to help in the understanding of an application's behavior while it is running. Using Sampler, you can see where an application spends most of its time executing. It can also summarize why certain allocation routines, system calls, or other functions were called. [Figure 2.22](#) shows Sampler in action sampling Safari, Apple's turbo Web browser.

Figure 2.22. The Sampler window.

debug and optimize them. [Figure 2.24](#) shows the USB Prober in action.

Figure 2.24. The USB Prober window.



The utilities listed here are only the ones supplied by Apple; dozens, if not hundreds, more are available from third-party developers as commercial software, shareware, or freeware. Many developers also write their own utilities and editors to use in their development efforts. The Online Resources listing in [Appendix B](#) is a good place to start your search for that perfect utility—and if it doesn't exist, create it! But wait... there's more!

Command-Line Developer Utilities

Apple also includes a collection of utilities that are only available from the command line. Some are standard to UNIX, and others are specific to Mac OS X. Many of these can be used to remotely debug an application via Telnet. Let's introduce them one by one—you'll know if you need them.

agvtool is the Apple generic versioning tool that is used to get or set the current version of a binary.

CpMac is used to copy Macintosh files. This utility copies both the resource and data forks.

DeRez decompiles resource files into Rez language files.

fs_usage logs all file operations including virtual memory paging for one or all processes.

GetFileInfo returns detailed file information about the selected file.

heap lists all heap memory in a process.

leaks searches a process's heap memory for unreferenced buffers, also known as leaks.

malloc_history displays a history of all heap allocations by a process.

MergePef allows you to merge Preferred Executable Format files.

MvMac is used to move Macintosh files. This utility moves both the resource and data forks.

ps lists all currently running processes.

ResMerger is a resource management tool that allows you, among other things, to copy resources from the resource fork into the data fork of another file and vice versa.

Rez compiles Rez language files into resource files.

RezWack allows you to convert a resource file into a format that can be transferred to a non-resource-fork-friendly platform.

sample can be used to profile a process at specific intervals, similar to the Sampler application listed earlier.

sc_usage logs all Mach system calls.

sdp converts XML files ending in .sdef into either .r files (for Carbon use) or .scriptSuite or .scriptTerminology files (for Cocoa use).

SetFile sets Finder information for Macintosh files.

SplitForks separates the resource and data forks of a file into a UNIX-friendly format.

top displays a list of currently running processes and their resource usage. This list is continually updated.

UnRezWack allows you to convert a RezWacked file back to a standard resource file.

vmmap displays detailed information of how virtual memory is being used including where malloc memory and libraries are stored.

WSMakeStubs generates static Objective-C stubs from WSDL files for use with WebServices

[\[Team LiB \]](#)

Frameworks

Similar to libraries, frameworks are collections of code or resources that together allow your programs to access certain features of the system and other software. By adding frameworks to your Project Builder projects and including the appropriate header files, you can begin using them immediately. A little documentation helps too.

The `/System/Library/Frameworks` directory contains dozens of frameworks—some documented, some not. Here is a partial list as of this writing with some explanations as to what each one does.

The **AddressBook.framework** is used to access items in the Address Book database that is new with Mac OS X Jaguar.

The **AppKit.framework** allows access to all AppKit Objective-C classes such as `NSApplication`, `NSButton`, `NSColor`, `NSDocument`, `NSImage`, `NSMenuItem`, `NSResponder`, `NSView`, and `NSWindow`. We will be using these classes throughout the book.

The **AppleScriptKit.framework** contains resources, AppleScript definition files, and event handlers for use when implementing AppleScript in your applications.

The **ApplicationServices.framework** contains multiple frameworks including those that support ColorSync, CoreGraphics, FindByContent, Language Analysis, and Speech Synthesis.

The **Carbon.framework** contains multiple frameworks including those that support Sound, Help, Human Interface Toolbox, HTML Rendering, Navigation Services, Printing, and the Security Human Interface.

The **Cocoa.framework** simply contains a header file that includes the AppKit and Foundation framework's header files for convenience.

The **CoreFoundation.framework** allows access to all CoreFoundation items such as `CFArray`, `CFBundle`, `CFDictionary`, `CFString`, and so on (see `Foundation.framework`).

The **CoreMIDI.framework** allows access to the MIDI (Musical Instrument Digital Interface) capabilities built into Mac OS X.

The **CoreServices.framework** contains multiple frameworks including those that support networking and Web services.

The **DiscRecording.framework** allows access to the CD (Compact Disc) and DVD (Digital Video Disc) disc burning capabilities that are built into Mac OS X.

The **Foundation.framework** allows access to all Foundation Objective-C classes such as `NSArray`, `NSBundle`, `NSDictionary`, `NSString`, and so on. We will be using these classes throughout the book (see `CoreFoundation.framework`).

The **InterfaceBuilder.framework** allows developers to build palettes and objects for use within the Interface Builder application.

The **IOBluetooth.framework** and **IOBluetoothUI.framework** allow developers to communicate with Bluetooth devices via capabilities built into Mac OS X.

The **IOKit.framework** is used to develop drivers for Mac OS X.

The **Message.framework** includes `NSMailDelivery`, which is a class that allows you to easily send an email message from within your Cocoa application.

The **PreferencePanes.framework** is used to implement Preference Panes, popularized by the System Preferences application. We will implement a Preference Pane in [Chapter 8](#).

The **ScreenSaver.framework** is used to implement screen savers, now called Screen Effects, under Mac OS X. We will implement a Screen Effect in [Chapter 10](#).

The **WebCore** framework is a Mac OS X specific version of the cross-platform KHTML library. Apple's Safari Web browser uses WebCore to implement its HTML parsing engine.

Be sure to peruse the Frameworks directory yourself for more goodies awaiting your discovery.

[\[Team LiB \]](#)

◀ PREVIOUS NEXT ▶

Conclusion

That ends our brief introduction to the tools you will be using while swimming in the sea of Mac OS X development. Although we will only use a few of these tools throughout the book, it is very important to know what is available to you as a developer. If you didn't know about the hammer, you might try to use your hand to pound in a nail—some of us already have! So, be sure to peruse the Developer directory to see what else is in there including scripts, templates, and examples.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Part II: Application Level

[3 Cocoa Applications](#)

[4 Cocoa Plug-ins](#)

[5 Carbon Plug-ins](#)

[6 Frameworks](#)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 3. Cocoa Applications

"Two nibs are better than one."

—Anonymous

Now that you know a bit about what makes Mac OS X tick, let's jump right in and create a document-based application using Cocoa. If you're reading this book, you might have created simple non-document-based applications in Cocoa. We'll be doing this throughout later chapters as test-bed programs are created. You know how simple they are to put together quickly. The document-based application, however, is a bit more complex. But once you see one examined, you will have little trouble creating your own.

Document-based applications allow a user to work with multiple documents at the same time. Whereas a non-document-based application usually has one window and can only manage one set of data at a time, document-based applications can have many windows open simultaneously—each accessing different source data. One window is always the focus of the user while the other windows are ready and waiting for input, sitting quietly in the background. When discussing sets of data, having the ability to open multiple views simultaneously gives you the ability to work in parallel on many tasks.

Note

When I refer to a set of data, this can mean file-based data sitting on your local hard disk or stream-based data that is transferred over a network, possibly from a database. A document-based approach can be used to edit either of these types of data.

Mail.app, Microsoft Word, Adobe Photoshop, Project Builder, and Interface Builder are all examples of applications that take a document-based approach. You can have multiple images open in Photoshop and edit between them, back and forth, as much as you like.

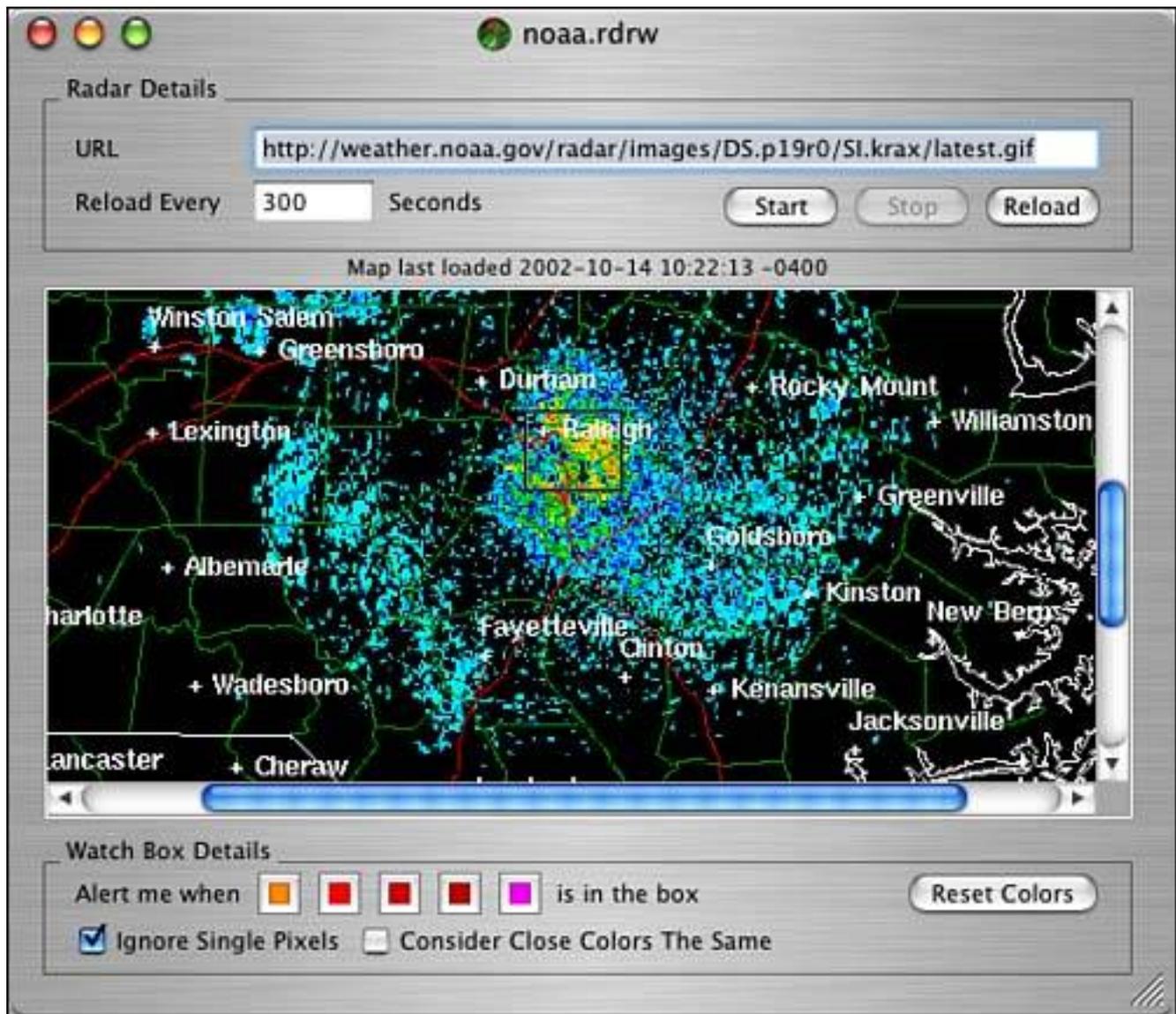
As of this writing, iTunes and iPhoto are both examples of applications that are not truly document based. That is, they can only operate on one item at a time. iTunes only plays one song at a time. IPhoto only allows you to edit one image at a time. When developers refer to document-based applications, they usually mean the ability to open multiple documents at once. Windows users know this as the multiple document interface (MDI).

The specific requirements of your application will determine whether it should be document-based or not. If your users want to work with more than one set of data, you probably want to use a document-based application. Many times, this will be obvious; sometimes it will not. If you're unsure, look for other applications that work similarly to yours and see how you might learn from their approaches. Also, be sure to talk to potential users of your application to see what they would expect from it.

The Result

Let's discuss the document-based application that we will be examining in this chapter. RadarWatcherX, shown in [Figure 3.1](#), allows you to create, open, and save multiple documents that repeatedly load a Doppler radar image and scan a "watch box" for specific colors in the image. This allows you to choose your local radar image from any weather-related Web site and keep an eye on a specific area within the image for storms or heavy rain that might be approaching. If the colors you choose appear in the watch box, an alarm will sound!

Figure 3.1. A RadarWatcherX document window.



Although you might not be able to tell from the screenshot, RadarWatcherX implements many common (and some not so common) tasks that you might need when building applications in Cocoa. Some of these include

- Document window management

- Saving and opening XML property lists (plists)
- Custom views
- Scrolling views
- Timers
- Internet connectivity
- Image manipulation
- Application preferences

If you are not familiar with some of these items, don't worry. I will discuss them enough here to get you started, and some (such as property lists) will be elaborated on in future chapters. Let's dive into the project file and source code and see how this project works.

Note

Property lists (plists) are text files that are formatted in either XML format or an older ASCII format. They are a great solution to store preference data because they support many different data types within the text. You can store numbers, strings, even raw data such as icons and pictures. You will see property lists used throughout your Cocoa development experience.

[\[Team LiB \]](#)

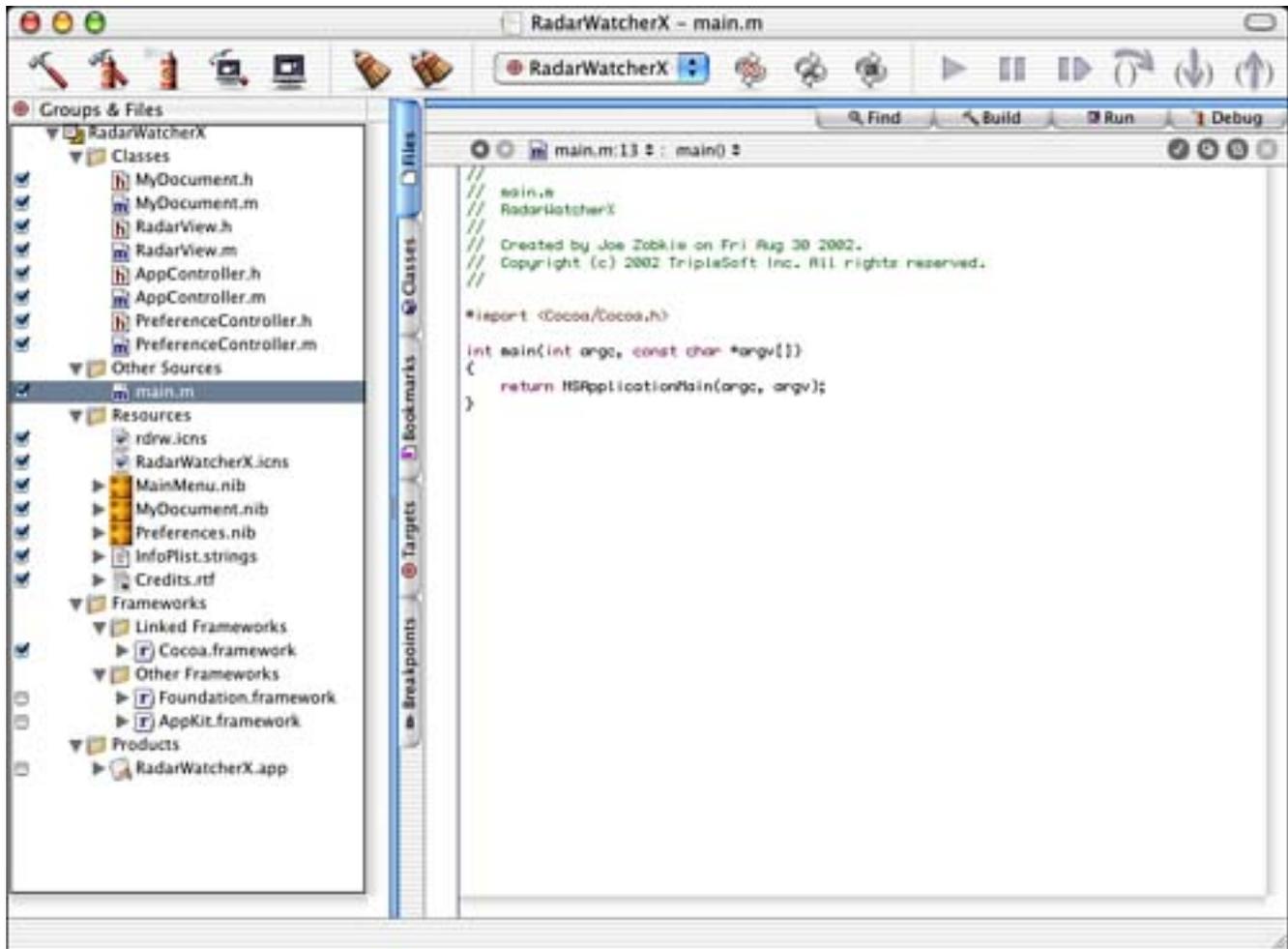
◀ PREVIOUS

NEXT ▶

The Project

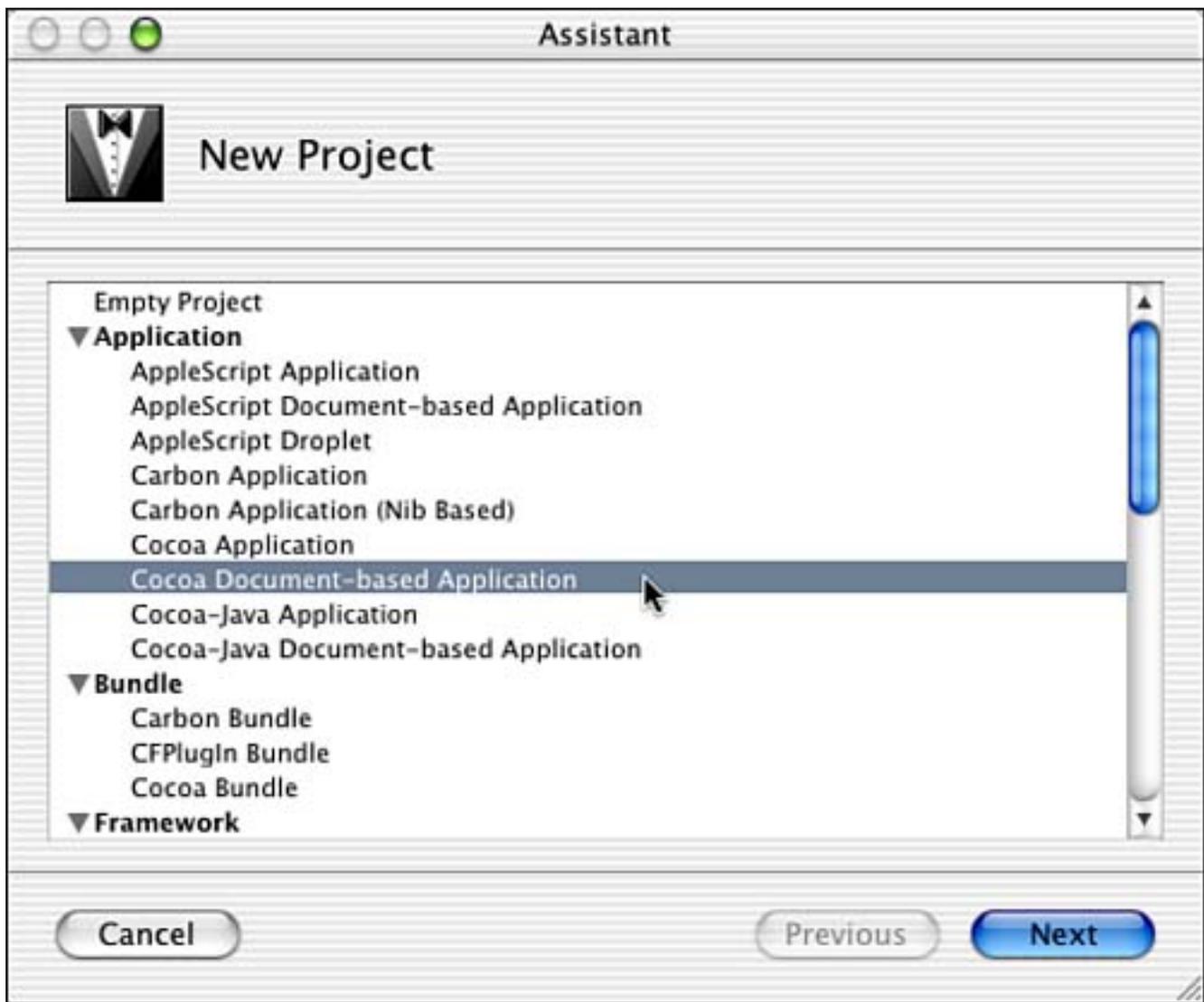
As you can see, this application project in [Figure 3.2](#) consists of a fair number of classes and resources. You should download the source code from the Web site for this book and follow along as we dissect the pieces. This is not a step-by-step recreation of the project, but rather a detailed discussion of the major pieces of the completed project.

Figure 3.2. The RadarWatcherX project in Project Builder.



When I began this as a new project, I made use of Project Builder's New Project Assistant, seen in [Figure 3.3](#), by selecting New Project from the File menu and selecting Cocoa Document-based application. The Assistant makes it easy to create a new project based on a template. There are templates for AppleScript, Carbon, and Cocoa applications as well as other more esoteric types. Use the templates; they can be a great time-saver and help you learn as you explore new aspects of Mac OS X programming.

Figure 3.3. The Project Builder New Project Assistant.



First Things First: Copyrights and Credits

The first thing I normally do in an application is take care of "fixing" all the little things that I know I'll have to do eventually. This includes editing the items in the `InfoPlist.strings` file, as well as adding a `Credits.rtf` file, if it does not already exist, to the project. The `InfoPlist.strings`, seen in [Figure 3.4](#), contains a few important localized strings such as the copyright and application version information. The optional `Credits.rtf` file, seen in [Figure 3.5](#), contains detailed information that appears in a scrolling list in the About Box, which indicates who did what and why on this project. You can create it with TextEdit and add it to the project as you would any other file. Once added, you can edit it directly in Project Builder. Both of these files are saved inside the application as part of its package.

Figure 3.4. InfoPlist.strings contents.



Figure 3.5. Credits.rtf contents.



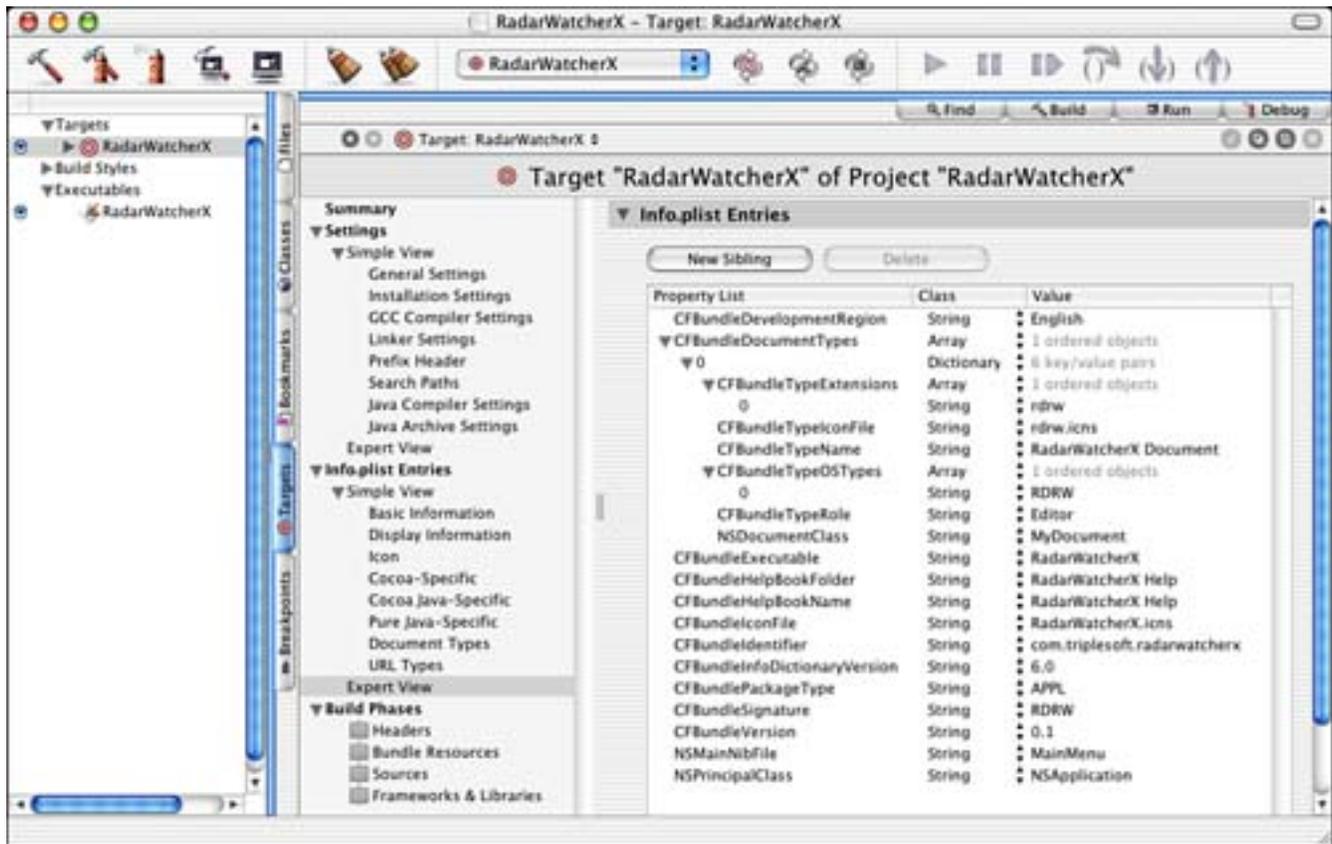
Note

These filenames are case sensitive as are many things in the UNIX environment on which Mac OS X is based. If you ever find that something isn't working right but you know a file is in the correct location, make sure to check the case. People have spent hours searching for bugs caused solely by the case of a single character.

First Things First: Other Info.plist Entries

Many other entries can be added to `Info.plist` that are not required to be localized. I find that the easiest way to manage these entries is by using Expert View in the Targets tab (see [Figure 3.6](#)). Entries in this list are in outline view format and contain such information as the document types supported by the application (including filename extensions), the help filename, the icon filename, the bundle identifier (a string that uniquely identifies this application), the main nib filename, and the name of the principal class in the application. Ultimately, these entries are stored in a property list file.

Figure 3.6. Info.plist entries.



Note

Although we are looking at Expert View, you can see from the screen shot that there is also a Simple View. Simple View's contextual layout might be easier on your eyes and brain. Click on Simple View or any one of its sub-views to see what it looks like.

One thing to note is that all of your projects that save default preferences (discussed later in this chapter) will need to have a unique `CFBundleIdentifier` entry. Add this key if there isn't already a placeholder for it. The convention is to use the reverse domain of your company with the name of the product at the end. In my case, TripleSoft Inc. uses the domain name `triplesoft.com`. RadarWatcherX is the name of my project. Therefore, my unique identifier is `com.triplesoft.radarwatcherx`. For your own projects, you should change the domain name accordingly. If you do not have a domain name, you can use something based on your email address, such as `com.mac.billjones.myappname`.

You can actually add your own custom keys to this list as well. Many times developers will place values in keys in this file that they then read in to their application at runtime. These values might or might not be editable by the user via the application's user interface. You can use `NSBundle`'s `-infoDictionary` method to access the contents of the `Info.plist` file. You might consider putting the names of preference files here or the name of a server that you use to check for the latest version of your software. Any value that you simply don't want to hard-code is a good candidate for `Info.plist`.

First Things First: Application and Document Icons

Next, you might choose to add an icon to your application and documents. Some folks might prefer to save this until the end of the development cycle; that's fine too. However, any application that you plan to release to the public should have a set of custom icons. This can be an interesting task to accomplish on your own for a variety of reasons. Let's discuss.

First, your icons need to look good these days. Before OS X existed, you could get away with an icon that didn't look all that professional. OS X and Aqua have changed the game, and many non-artistic types have not been able to keep up. If you make an icon, be sure to make it look like it belongs in OS X and not System 3.2. If you are an incredibly talented programmer but your sister got the artistic skills, let her make the icon.

A few applications are available that can help you on your quest to make decent looking icons. There might be more, but these are the best I've found:

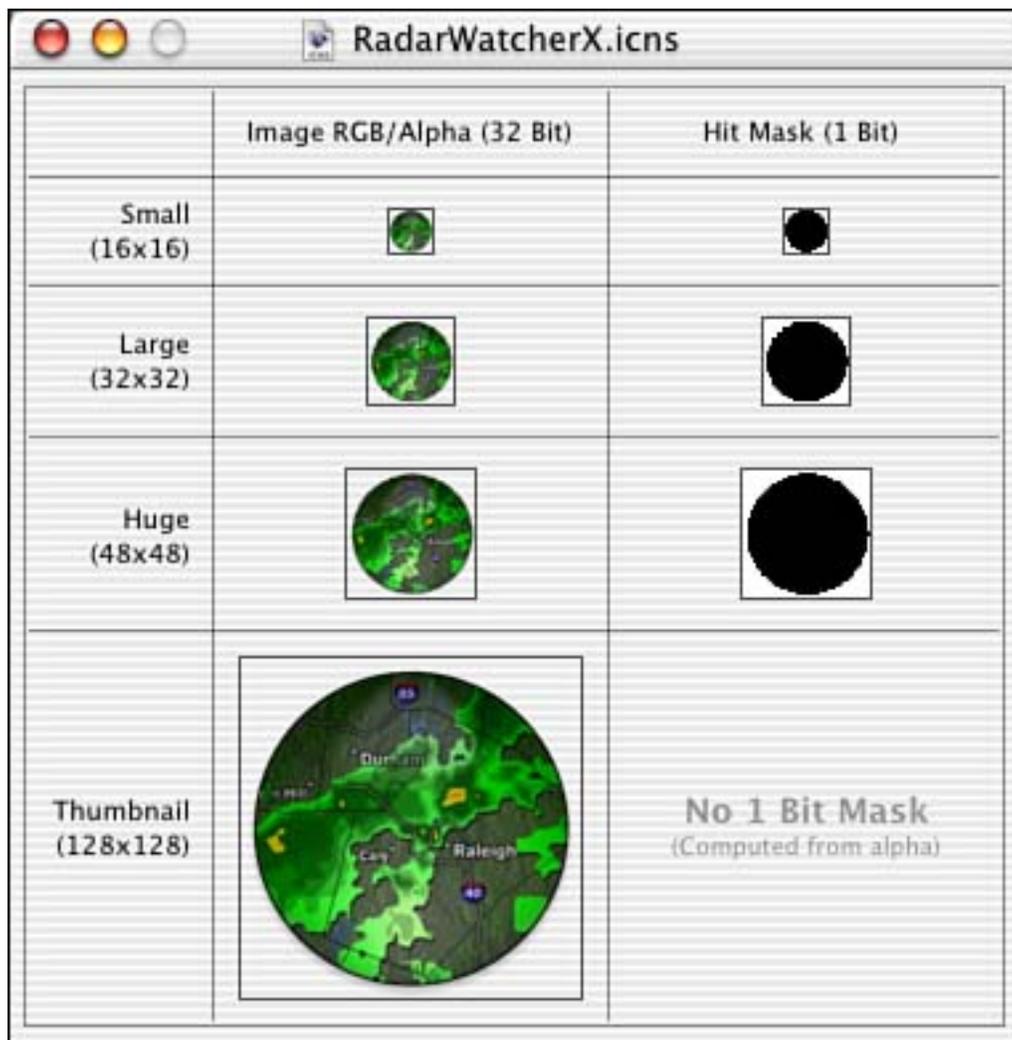
- Adobe Photoshop or Photoshop Elements (<http://www.adobe.com/>) are excellent applications for high-end image manipulation, but you still need some talent.
- Stick Software's AquaTint (<http://www.sticksoftware.com>) assists you in creating glossy, liquid Aqua-style images with ease. AquaTint comes with very helpful documentation that can assist you in making great looking icons. Remember, the Alpha Channel is your friend when creating OS X icons.
- Infinity-to-the-Power-of-Infinity's Can Combine Icons (<http://www.ittpoi.com/>) is another excellent application that makes it easy to layer precreated images together to make eye-catching (and meaningful) icons. Using this in conjunction with AquaTint can be a winning proposition.
- Icon Factory (<http://www.iconfactory.com/>) is a great Web site that offers thousands of icons for use on your Macintosh. You will need to check with the author of any icons before you use them with your application, but this is a good place to locate talented artists who might help you on the icon creation side of the equation.

Note

Apple Computer also provides some great information on Aqua icon design philosophy. Visit <http://developer.apple.com/ue/aqua/icons.html> for more information.

Ultimately, you will use the IconComposer application, included with the OS X development tools and seen in [Figure 3.7](#), to create a file of type icns. This file contains all the data for any particular icon. In this case, I have created two of these files—one for the application icon and one for the document icon.

Figure 3.7. IconComposer window showing the application icon.



After you have the icon files created, you can easily add them to the project and add the appropriate **Info.plist** entries as shown previously. In this case, our **CFBundleIconFile** is set to the filename of our application icon: **RadarWatcherX.icns**.

The document icon configuration is a bit more complex. We need to fill in the **CFBundleDocumentTypes** array with information on the file extensions and icon filenames. Our documents end in **.rdrw** as shown in the **CFBundleTypeExtensions** array, and our icon file for that file type is **rdrw.icns** as shown in the **CFBundleTypeIconFile** entry. We also need to fill in **CFBundleTypeName**, **CFBundleTypeOSTypes**, **CFBundleTypeRole**, and **NSDocumentClass** of our document. RadarWatcherX only supports one document type, but it could easily manage multiple document types with multiple icons simply by adding the proper additional array items.

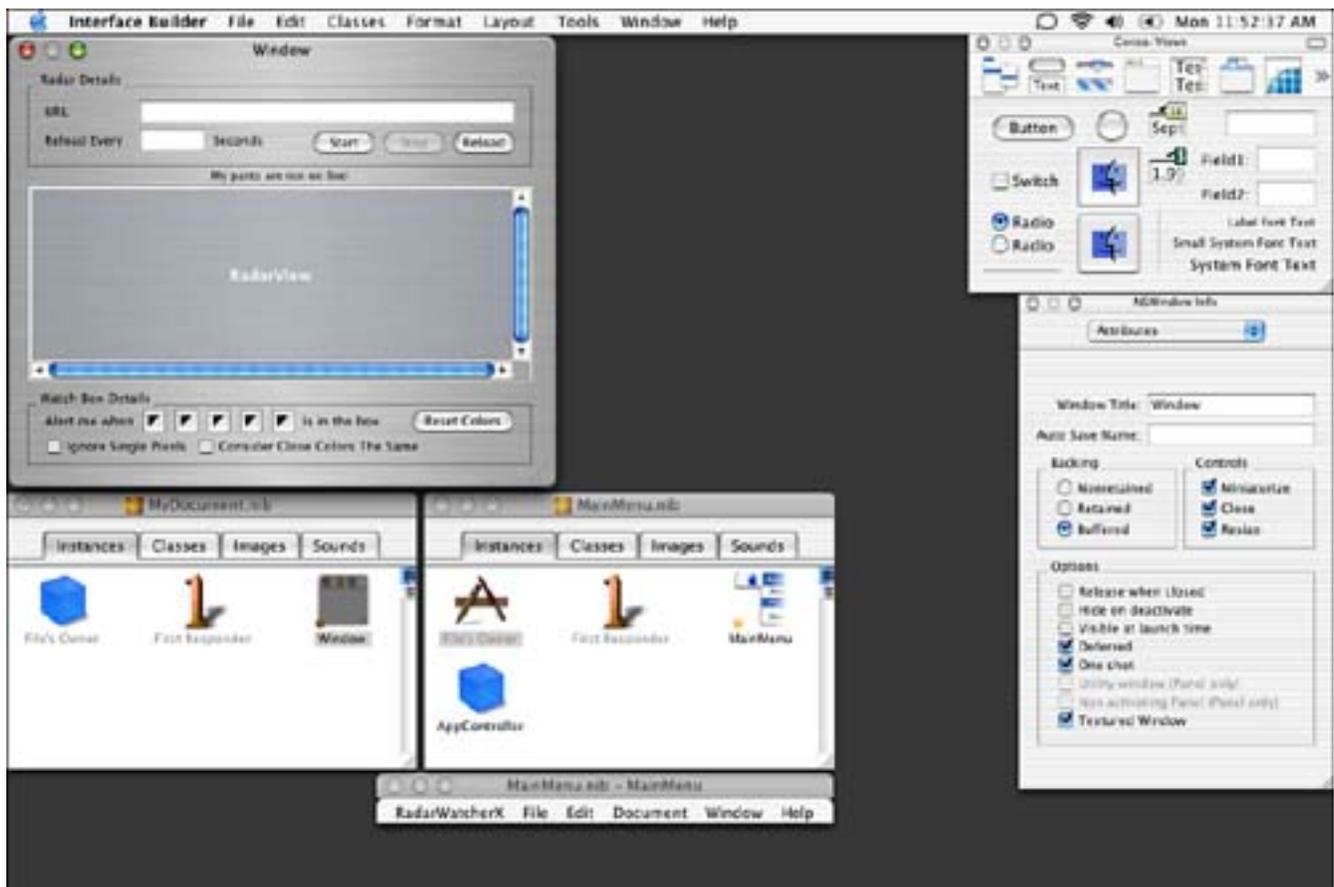
Note

Whenever you alter the icons of an application, they might not always "take" right away. That is, if you create an icon, set up all the **Info.plist** entries properly, the icons appear, and then you decide to tweak the image of the icon, you might not see it in the Finder right away. Sometimes you need to log out and log back in for the changes to take effect. The Finder caches icons and doesn't always notice that a change has occurred. This can happen with other data within your application as well, such as version information, and so on. If things don't seem right, try logging out, restarting, or copying the application to another computer to verify that it is just your environment and not your project itself causing the trouble. Restart the computer as a last resort.

Interface Builder

The next thing I usually jump on in a project is the user interface. Assuming that you've planned your application well enough, you probably know what the menus and document window will look like. The first thing you will notice when working with a document-based Cocoa application is that you have two nib files: **MainMenu.nib** for the application resources and **MyDocument.nib** for the document resources; both are shown in [Figure 3.8](#).

Figure 3.8. RadarWatcherX in Interface Builder.



MainMenu.nib contains the main application menus and any instances of classes that need to work at the application level. This is where you would put your **AppController**, for instance. This allows you to easily make it the delegate of the **NSApplication** object, which is the File's Owner. You can also set up your menus in this file and link them to actions that you create in the First Responder. Menu items are usually connected to the First Responder, so the actions can be routed properly to any object that responds to them—in our case, our document object.

You can create any number of single-window applications without ever having to worry about File's Owner or First Responder—you simply create controllers that span the life of the program, and all is well. However, when you create document-based applications, these items become a lot more important. The File's Owner, for example, represents what is essentially the nib's default controller. Items in the nib are usually hooked up directly to the File's Owner's **IBOutlet**s.

`MyDocument.nib` contains our main document window that is linked to actions and outlets in the File's Owner, the `MyDocument` class. Our documents only have one window attached to them, but you can have documents with multiple windows if you so desire. Our document window also contains a custom view that we will discuss later, `RadarView`.

Note

If you do create a document-based application in which the documents have multiple windows associated with them, you will want to create a separate nib file for each window. Each nib file will be owned by an `NSWindowController` subclass. For example, a document with a front view window and a back view window would include two `NSWindowControllers`—one for each. There would also be two nib files—one for each. Each `NSWindowController` would be the File's Owner in one of the nib files. The `NSDocument` can then manage the multiple controllers for you. This approach will allow you to fully leverage the power of Cocoa's document-based application model.

There are multiple ways to manage your nib files, source files, and the connections between them. You can create your source files in Project Builder and drag the header files to Interface Builder to have it update itself with the outlets and actions that you've defined in the class. You can also create the source files from within Interface Builder using items in the Classes menu and have them automatically added to the project. In most cases, I initially create the files in Interface Builder and then do further editing in Project Builder, dragging the header file to Interface Builder each time I make a change so that everything is in sync. As you tweak your user interface, you might find this approach to be useful as well. Maybe someday, Interface Builder and Project Builder will be one application, but for now this is the way it works.

Main

The function "main" of any Cocoa application, document-based or otherwise, simply contains a call to the `NSApplicationMain` function (see [Listing 3.1](#)). This function initializes your application and kicks off the main event loop. You need not worry about anything beyond this unless your application has special needs. Project Builder automatically generates this file for you when the project is created.

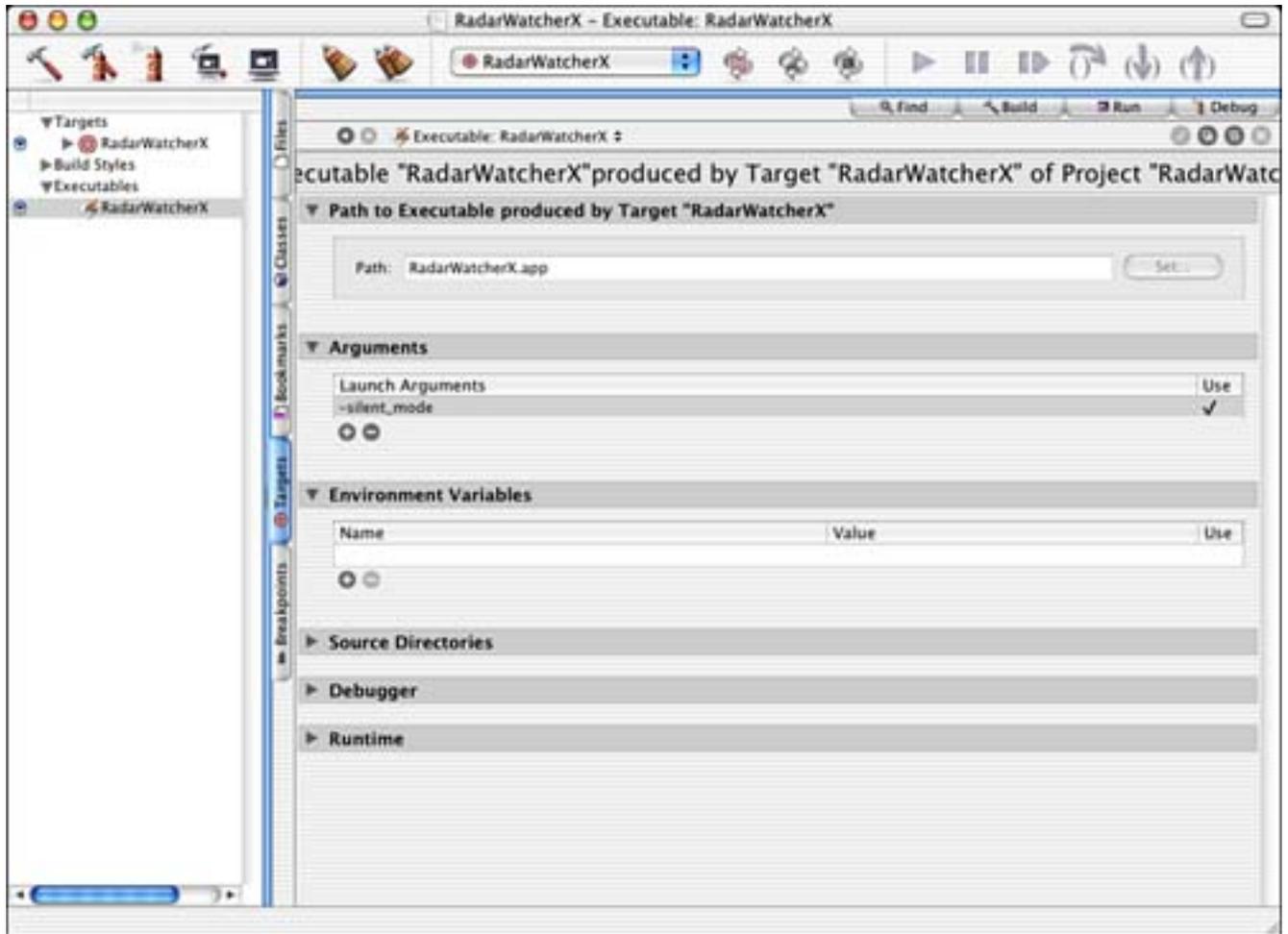
Listing 3.1 Main.m

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

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

Although `RadarWatcherX` doesn't make use of this feature, you can set up your application to accept arguments when it is launched. By selecting Edit Active Executable from the Project menu in Project Builder, you can easily add environment variables and arguments. You can process these arguments in main or by using `NSProcessInfo`'s `arguments` method to examine the arguments later on. Although these arguments won't be used when your application is launched from the Finder, you can have Project Builder pass them when it launches your application directly, as seen in [Figure 3.9](#).

Figure 3.9. Editing the RadarWatcherX executable.



MyDocument

The `NSDocument` subclass `MyDocument` is the centerpiece of our project. This class handles everything in our document from saving and opening RadarWatcherX document files, keeping track of changes to the document, and managing the automatic reloading of the radar image. Let's look at the functions in `MyDocument`.

Storing Document Data via MutableDictionary

The first thing you will notice in the `MyDocument.m` source file is a list of global `NSStrings` (see [Listing 3.2](#)). These are used to represent the key values of the key/data pairs in the property list-based document files that `MyDocument` manages. Essentially, each user interface element in the document window has a key value associated with it to allow easy saving and retrieving of the value. `MyDocument.h` contains the associated extern declarations.

Listing 3.2 `NSMutableDictionary` Keys

```
NSString *keyURL = @"keyURL";
NSString *keyReloadSeconds = @"keyReloadSeconds";
NSString *keyWatchBoxRect = @"keyWatchBoxRect";
NSString *keyColors[kNumColors] = {"keyColors1", @"keyColors2", @"keyColors3",
```

```

        @"keyCol ors4", @"keyCol ors5"};
NSString *keyIgnoreSi ngl ePi xel s = @"keyI gnoreSi ngl ePi xel s";
NSString *keyCl oseCol ors = @"keyCl oseCol ors";

```

The `-init` override method of `MyDocument` in [Listing 3.3](#) creates the `NSMutableDictionary` instance variable that is used to store the current key/value pairs for the document. The entries are then initialized. Note how you can easily add numerous types of objects to a dictionary by calling the dictionary's `-setObject: forKey:` method. Dictionaries are a great way to store hierarchical or flat data that needs to be represented by a key. After this code is executed, the dictionary is filled with valid default values for the document. Later in the `-dealloc` override, called when the document is closed, we will release the `NSMutableDictionary` and set it to `nil`.

Listing 3.3 `MyDocument -init`

```

m_md = [NSMutableDictionary dictionary] retain];
[m_md setObject:kDefaultURL forKey:keyURL];
[m_md setObject:[NSNumber numberWithInt:kDefaultReloadSeconds]
    forKey:keyReloadSeconds];
[m_md setObject:[NSStringFromRect(NSMakeRect(0, 0, 0, 0))
    forKey:keyWatchBoxRect];
[m_md setObject:[NSArchiver archivedDataWithRootObject:[NSColor clearColor]]
    forKey:keyCol ors[0]];
[m_md setObject:[NSArchiver archivedDataWithRootObject:[NSColor clearColor]]
    forKey:keyCol ors[1]];
[m_md setObject:[NSArchiver archivedDataWithRootObject:[NSColor clearColor]]
    forKey:keyCol ors[2]];
[m_md setObject:[NSArchiver archivedDataWithRootObject:[NSColor clearColor]]
    forKey:keyCol ors[3]];
[m_md setObject:[NSArchiver archivedDataWithRootObject:[NSColor clearColor]]
    forKey:keyCol ors[4]];
[m_md setObject:[NSNumber numberWithInt:NSOffState]
    forKey:keyIgnoreSi ngl ePi xel s];
[m_md setObject:[NSNumber numberWithInt:NSOffState]
    forKey:keyCl oseCol ors];

```

Note

There are so many styles, but so little time. Through my years of programming, I've developed my own style, as you might have as well. When I first came to Objective-C, I noticed that a lot of the sample code I saw made it difficult to distinguish instance variables from local variables. To remedy this, I tend to use an `m_` before my instance variable names. The `m` comes from the C++ term "member variable." I've seen others use only an `_`. Any way you choose to do it, if you see an `m_` in this book, you can be sure that you are looking at an instance variable.

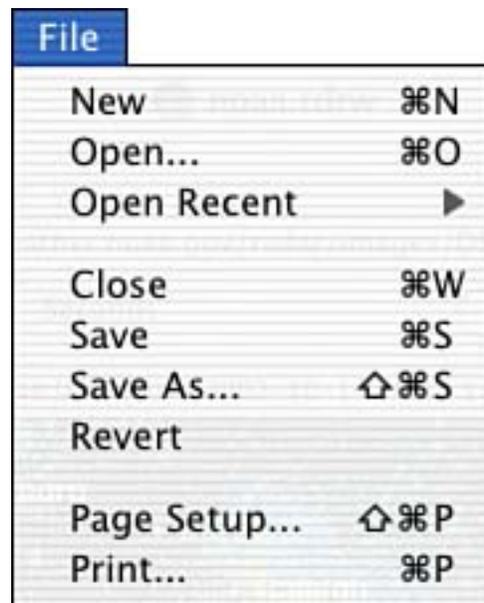
Reading and Writing Document Data via `MutableDictionary`

There are numerous ways to save data using a document-based application. In our case we are using an `NSMutableDictionary` and the `NSDocument -writeToFile: ofType:` and `-readFromFile: ofType:` methods. This makes a very straightforward approach to manage our data. Simply implementing these two

functions handles all the file-related items in the File menu. This is a very nice feature of the Cocoa framework.

In order to save to a file, we override the `-writeToFile:ofType:` method of `NSDocument` as shown in [Listing 3.4](#). This method takes a filename and a file type as arguments and returns a `BOOL YES` upon success. Because our document only saves one file type, we essentially ignore the file type argument. Before we can write our `NSMutableDictionary` to the file on disk, however, we need to ensure that it is up-to-date with the current settings of the controls on the screen.

Figure 3.10. The RadarWatcherX File menu.



We use various methods to create data in the proper format to pass to the `-setObject:forKey:` method of the `NSMutableDictionary`. For example, we want our rectangle to be a string format so that we can easily see the values in the file. As of this writing, `NSColorWell` objects don't know how to write themselves to property lists, so we archive them as raw data. We also retrieve the current state of the check boxes. Other user interface items, such as the text fields, are already up-to-date via the `NSNotification` notification that is sent to `MyDocument` each time the user types them in. The last thing we need to do is tell the `NSMutableDictionary` to `-writeToFile:atomically:`. That's all there is to it!

Listing 3.4 `MyDocument -writeToFile:ofType:`

```
// NSRect are best off saved as string representations
// so the developer can edit them
[m_md setObject: NSStringFromRect([m_radarView watchBoxRect])
    forKey: keyWatchBoxRect];

// NSColor should save but do not, so we must archive
// and unarchive them to get them to work
[m_md setObject: [NSArchiver archivedDataWithRootObject: [m_colorWell1 color]]
    forKey: keyColors[0]];
[m_md setObject: [NSArchiver archivedDataWithRootObject: [m_colorWell2 color]]
    forKey: keyColors[1]];
[m_md setObject: [NSArchiver archivedDataWithRootObject: [m_colorWell3 color]]
    forKey: keyColors[2]];
```

```

[m_md setObject:[NSArchiver archivedDataWithRootObject:[m_colorWell4 color]]
 forKey:keyColors[3]];
[m_md setObject:[NSArchiver archivedDataWithRootObject:[m_colorWell5 color]]
 forKey:keyColors[4]];

// Retrieve the state of the checkbox
[m_md setObject:[NSNumber numberWithInt:[m_ignoreSinglePixelsButton state]]
 forKey:keyIgnoreSinglePixels];
[m_md setObject:[NSNumber numberWithInt:[m_closeColorsButton state]]
 forKey:keyCloseColors];

// Write the current dictionary to the file
return [m_md writeToFile:fileName atomically:YES];

```

Note

Atomically? When writing to a file atomically, you are telling the dictionary to write to a backup file first; then if no errors have occurred, rename the backup file to the specified filename. Otherwise, the file is written directly to the specified filename.

Now, to read the data back, we override the `-readFromFile:ofType:` method of `NSDocument` as shown in [Listing 3.5](#). This method takes the same arguments as `-writeToFile:ofType:`. We simply release our current dictionary and load the dictionary from the file using the `+dictionaryWithContentsOfFile:` method of `NSMutableDictionary`.

Listing 3.5 MyDocument -readFromFile:ofType:

```

// Release the current dictionary
[m_md release];
m_md = nil;

// Load the new data from the file
m_md = [[NSMutableDictionary dictionaryWithContentsOfFile:fileName] retain];
if (m_md) {
    // Update the UI in case this is a Revert
    [self updateUI];

    // Return a positive result
    return YES;
}

return NO; // Failure

```

If it loaded properly, we update our user interface by calling `-updateUI`, shown in [Listing 3.6](#), which handles converting the archived data back to a format we can use. That is, for example, for every call to `NSArchiver`, we must have a balanced call to `NSUnarchiver`.

Listing 3.6 MyDocument -updateUI

```

// Update the UI with the data from the latest file or revert action
[m_urlTextField setStringValue: [m_md objectForKey: keyURL]];
[m_reloadSecondsTextField setStringValue: [m_md objectForKey: keyReloadSeconds]];

// NSColor should save but do not, so we must archive
// and unarchive them to get them to work
[m_colorWell1 setColor: [NSUnarchiveObjectWithData:
    [m_md objectForKey: keyColors[0]]]];
[m_colorWell2 setColor: [NSUnarchiveObjectWithData:
    [m_md objectForKey: keyColors[1]]]];
[m_colorWell3 setColor: [NSUnarchiveObjectWithData:
    [m_md objectForKey: keyColors[2]]]];
[m_colorWell4 setColor: [NSUnarchiveObjectWithData:
    [m_md objectForKey: keyColors[3]]]];
[m_colorWell5 setColor: [NSUnarchiveObjectWithData:
    [m_md objectForKey: keyColors[4]]]];

// Set the state of the checkbox
[m_ignoreSignalPixelsButton setState: [[m_md objectForKey: keyIgnoreSignalPixels]
    intValue]];
[m_closeColorsButton setState: [[m_md objectForKey: keyCloseColors]
    intValue]];

// Tell the radar view what the current watch box
// rectangle is so it can draw it properly
[m_radarView setWatchBoxRect: NSRectFromString([m_md objectForKey:
    keyWatchBoxRect])];

```

One thing to note is that the `-readFromFile:ofType:` method is not only called when a document file is opened, but also when the user selects Revert or Open Recent.

Now that you know how to read and write the data to the disk, let's see what it looks like saved in the file. [Listing 3.7](#) shows the property list file that is saved by the routines listed previously. Although the file ends in `.rdrw`, it is really just a text file. You can open it in any text editor. Note that the colors are "archived" as raw data, whereas the watch box rectangle can be edited by hand if you like.

Listing 3.7 RadarWatcherX .rdrw Data File

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>keyCloseColors</key>
    <integer>0</integer>
    <key>keyColors1</key>
    <data>
    BA0eXBI ZHN0cmVhbYED6I QBQI SEhAdOUONvbG9yAI SECE5TT2JqZWNOAI WEAWMBhARm
    ZmZmAYM/EJCRAAGG
    </data>
    <key>keyColors2</key>
    <data>
    BA0eXBI ZHN0cmVhbYED6I QBQI SEhAdOUONvbG9yAI SECE5TT2JqZWNOAI WEAWMBhARm
    ZmZmAQAAAYY=

```

```

</dat a>
<key>keyCol ors3</key>
<dat a>
BA tOeXBI ZHN0cmVhbYED6I QBQI SEhAdOUONvbG9yAI SECE5TT2JqZWNOAI WEAWMBhARm
ZmZmgz9W1t cAAAGG
</dat a>
<key>keyCol ors4</key>
<dat a>
BA tOeXBI ZHN0cmVhbYED6I QBQI SEhAdOUONvbG9yAI SECE5TT2JqZWNOAI WEAWMBhARm
ZmZmgz9AwMEAAAGG
</dat a>
<key>keyCol ors5</key>
<dat a>
BA tOeXBI ZHN0cmVhbYED6I QBQI SEhAdOUONvbG9yAI SECE5TT2JqZWNOAI WEAWMBhARm
ZmZmAQABAYY=
</dat a>
<key>keyI gnoreSi ngl ePi xel s</key>
<i nt eger>1</i nt eger>
<key>keyRel oadSeconds</key>
<i nt eger>300</i nt eger>
<key>keyURL</key>
<stri ng>http://weather.noaa.gov/radar/i mages/SI.krax/l at est. gif</stri ng>
<key>keyWat chBoxRect</key>
<stri ng>{{297, 304}, {46, 38}}</stri ng>
</di ct>
</pl i st>

```

Tracking Changes

Another important task that the document must tend to is tracking changes to itself. Whenever you open a document, it is considered clean. As soon as you make a change, it becomes dirty. If the user attempts to close a dirty document (or quit the application while a dirty document is open), he is prompted to save the document. It is up to the document itself to track its clean/dirty state.

You can get rather complex in tracking these changes, but we do it rather simply in RadarWatcherX. Essentially, whenever the user makes a change to a user interface component, we call the `NSDocument` method `-updateChangeCount:` passing `NSChangeDone`. This increments the change count of the document telling us it is dirty. As long as we call this method each time a change is made, `NSDocument` will automatically handle asking the user to save if needed.

Timer

`MyDocument` uses the value in the `keyReloadSeconds` key to set a timer. This timer is created and set when the user clicks the Start button and the subsequent `-startRadar:` action is called. We use the `NSTimer +scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` class method to create a repeating timer (see [Listing 3.8](#)). The timer takes an `NSTimeInterval` to know how often to fire, which object contains the method to execute, and the selector (method) within the target to call. In this case, the `-refreshRadar:` method is called when the timer fires. The timer is destroyed by invalidating and releasing the timer instance variable when the user presses the Stop button.

Listing 3.8 Creating a Timer

```
m_timer = [[NSTimer scheduledTimerWithTimeInterval:
    (NSTimeInterval) [[m_md objectForKey:keyReloadSeconds] doubleValue]
    target:self selector:@selector(refreshRadar:) userInfo:nil repeats: YES]
retain];
```

refreshRadar

The `-refreshRadar:` method in [Listings 3.9](#) through [3.11](#) is where the most important work of RadarWatcherX occurs. This is where the radar image is loaded and analyzed for colors. It is not only called from the timer, but also when the user presses the Reload button. Let's look at what is going on in this function.

Listing 3.9 MyDocument -refreshRadar: Image Loading

```
// Load the current image from the URL and display
// it on the screen, releasing any previous image
// Note we must use this method and not [UIImage
// initWithContentsOfURL] because it uses the cache
NSData *data = [[NSURL URLWithString: [m_md objectForKey:keyURL]]
    resourceDataUsingCache:NO];
UIImage *image = [[UIImage alloc] initWithData:data];
```

Before anything, we attempt to load the current URL typed into the URL field. Using `+ [NSURL URLWithString:]` and `- [NSURL resourceDataUsingCache:NO]` we force the image to be loaded from the server and not from our local cache. Given the data we then attempt to initialize an `UIImage` from it. Because `UIImage` understands a variety of image formats we can easily load JPEG, GIF or PNG files, to name a few.

Next, if we were able to create the image, we pass it immediately to our `RadarView`, as shown in [Listing 3.10](#). We will look at `RadarView` in more detail in a moment, but for now simply know that `RadarView` is a subclass of `NSView` and is the "owner" of the image that is displayed on the screen. `MyDocument` simply manages the means to obtain the image at regular intervals. `RadarView` is the class that handles the details of manipulating the image itself. Once we pass the image to `RadarView`, which retains it, we release it. We also update the user interface to display the current time as the "Map last loaded" time.

Listing 3.10 MyDocument -refreshRadar: Image Loaded

```
// If we loaded an image...
if (image) {

    // Pass it to the RadarView for display
    [m_radarView setImage:image];

    // We no longer need the image, m_radarView
    // has "retained" it at this point
    [image release];

    // Update the current time as the last time we loaded the image
    [m_statusTextField setValue:[NSString stringWithFormat:
        @"Map last loaded %@", [NSDate date]]];
```

Note

Speaking of `-retain` and `-release`, this is one of the big gotchas in Cocoa programming. It's easy enough to understand that if a block of memory is being used by one object, you don't want another object to free it. This is the concept of retain and release. If I retain an object, I am essentially increasing the `-retainCount` of that object by 1. Releasing an object lowers the `-retainCount` by 1. When the `-retainCount` reaches 0, the object is freed and is no longer valid.

The problem is that it can be difficult to know when you "obtain" an object if it has already been retained, or if you should retain it. If you retain it, you should also release it; but should you release it if you did not retain it?

The simple rule of thumb is that if you explicitly `+alloc` and `-init` an object, you should retain and release it as appropriate. If you obtain an object from another mechanism, it is probably created using `-autorelease`. This means that if you do not explicitly retain it, it will be released automatically at the end of the event loop. This is made possible by something called an autorelease pool, which is discussed in subsequent chapters. Autorelease pools make it easy for us to use `NSString` class methods in a function without having to worry much about managing the memory that `NSString` uses. However, if no autorelease pool is being used, you should consider your memory use carefully.

Now that `RadarView` has the image and has displayed it on the screen, we can check it for the colors we are concerned about. See [Listing 3.11](#). Essentially, we ask `RadarView` to see if the given color is within the current watch box, taking into account the `ignoreSinglePixels` and `closeColorState` options. Because we have five `NSColorWells`, we call the `RadarView` five times—each with a different color. If `RadarView` returns that the color was found in the watch box, we alert the user.

Listing 3.11 MyDocument `-refreshRadar`: Image Checking

```
// Check for colors
BOOL ignoreSinglePixelsState =
    ([m_ignoreSinglePixelsButton state] == NSOnState);
BOOL closeColorsState = ([m_closeColorsButton state] == NSOnState);
if ((([m_colorWell1 color] alphaComponent] > 0) &&
    ([m_radarView isColorInImageInRect: [m_colorWell1 color]
 ignoreSinglePixels: ignoreSinglePixelsState
 closeColors: closeColorsState] == YES)) {
    [self displayAlert:@"Found Color 1!"
 msg:@"A color was found in the watch box."
 defaultButton:@"OK"];
}
if ((([m_colorWell2 color] alphaComponent] > 0) &&
    ([m_radarView isColorInImageInRect: [m_colorWell2 color]
 ignoreSinglePixels: ignoreSinglePixelsState
 closeColors: closeColorsState] == YES)) {
    [self displayAlert:@"Found Color 2!"
 msg:@"A color was found in the watch box."
 defaultButton:@"OK"];
}
```

```

if ([[m_colorWell3 color] alphaComponent] > 0) &&
    ([m_radarView isColorInImageInRect: [m_colorWell3 color]
 ignoreSinglePixels: ignoreSinglePixelsState
 closeColors: closeColorsState] == YES)) {
    [self displayAlert:@"Found Color 3!"
     msg:@"A color was found in the watch box."
     defaultButton:@"OK"];
}
if ([[m_colorWell4 color] alphaComponent] > 0) &&
    ([m_radarView isColorInImageInRect: [m_colorWell4 color]
 ignoreSinglePixels: ignoreSinglePixelsState
 closeColors: closeColorsState] == YES)) {
    [self displayAlert:@"Found Color 4!"
     msg:@"A color was found in the watch box."
     defaultButton:@"OK"];
}
if ([[m_colorWell5 color] alphaComponent] > 0) &&
    ([m_radarView isColorInImageInRect: [m_colorWell5 color]
 ignoreSinglePixels: ignoreSinglePixelsState
 closeColors: closeColorsState] == YES)) {
    [self displayAlert:@"Found Color 5!"
     msg:@"A color was found in the watch box."
     defaultButton:@"OK"];
}
} else {
    NSBeep();
}

```

For now, this summarizes `MyDocument`'s responsibilities. Basically, it exists to manage the user interface and the document data. You could easily break out the user interface management functions into another window controller object if you felt it necessary. If your program needed more than one window per document, you would do this. Also, if your document class became too big, you might consider it as well. There is always more than one way to implement a task, especially in Cocoa! Peruse the code for more details and error checking code.

RadarView

Let's take a look at `RadarView`. As you saw in [Figure 3.8](#), `RadarView` is a Custom View in Interface Builder that is of type `RadarView`. When the nib file is loaded, a `RadarView` is automatically instantiated for us. The most important instance variables of `RadarView` are the radar image itself, which is passed in by `MyDocument`, and the watch box rectangle, which is drawn by the user and completely managed by `RadarView`. Placing the `RadarView` in an `NSScrollView` in Interface Builder makes it scrollable if the image is larger than the view.

Note

Something you will notice right away is that when a new image is loaded in a `RadarWatcherX` window, the scroll view scrolls in such a way that you see the lower-left corner of the image as the anchor point. The reason for this is that Mac OS X has changed the positioning of the world for "old school" Macintosh developers. It used to be that the upper-left corner was 0, 0 in the coordinate system of an image or view. Mac OS X changes this so that the lower-left corner is now 0, 0. There are ways of altering this for any particular view if you need to, but the odds are once you get used to it, you'll rarely have a need to.

Watch Box

By overriding `NSView`'s `-mouseDown:`, `-mouseDragged:`, and `-mouseUp:` methods, in [Listing 3.12](#), `RadarView` can completely manage the drawing and tracking of the watch box. Essentially, when the mouse button is clicked in the view, the down point is saved. As the mouse cursor is dragged, the current point is saved and the rectangle is drawn in the `-drawRect:` method. When the mouse button is released, the final point is saved and the final rectangle drawn.

Listing 3.12 `RadarView` Mouse Overrides

```
- (void)mouseDown: (NSEvent *) event
{
    if (m_image && !m_locked) {
        NSPoint p = [event locationInWindow];
        m_downPoint = [self convertPoint:p fromView:nil];
        m_currentPoint = m_downPoint;
        [self setNeedsDisplay:YES];
        [m_document updateChangeCount:NSChangeDone];
    }
}

- (void)mouseDragged: (NSEvent *) event
{
    if (m_image && !m_locked) {
        NSPoint p = [event locationInWindow];
        m_currentPoint = [self convertPoint:p fromView:nil];
        [[self superview] autoscroll:event];
        [self setNeedsDisplay:YES];
    }
}

- (void)mouseUp: (NSEvent *) event
{
    if (m_image && !m_locked) {
        NSPoint p = [event locationInWindow];
        m_currentPoint = [self convertPoint:p fromView:nil];
        [self setNeedsDisplay:YES];
    }
}
```

Checking for Colors

When `MyDocument` asks `RadarView` to check for colors in [Listing 3.11](#), the `-ignoreSinglePixels:closeColors:` method does the work. This method walks each pixel in the watch box in the image and compares it against the color passed from `MyDocument`.

If `ignoreSinglePixels` is `YES`, more than one of a color must be in the image in order for it to be considered "found." The problem is that some radar images have "blips" or artifacts that can mislead `RadarWatcherX`. If there is only one red pixel in a watch box, it probably isn't a storm, but just some sort of

ground clutter falsely interpreted by the radar itself.

If `closeColors` is `YES`, colors match if their hue component is within 10 degrees and their brightness component is within 20%. These are arbitrary values that I found worked well with the maps I was working with. The idea here is that some maps don't use a specific orange—for example, they might use a variety of oranges. This helps to alleviate the problem in which you are looking for an orange that is slightly off from the actually orange used in the map.

Listing 3.13 `RadarView - isColorInImageInRect: ignoreSinglePixels: closeColors:`

```
- (BOOL) isColorInImageInRect: (NSColor *) color
  ignoreSinglePixels: (BOOL) ignore
  closeColors: (BOOL) close
{
    int      x, y;
    long     found = 0;

    // Lock the image focus so we can perform direct pixel reads
    [m_image lockFocus];

    // Update the watch box rect
    m_watchBoxRect = [self currentRect];

    for (x = m_watchBoxRect.origin.x;
         x < m_watchBoxRect.origin.x + m_watchBoxRect.size.width; ++x) {
        for (y = m_watchBoxRect.origin.y;
             y < m_watchBoxRect.origin.y + m_watchBoxRect.size.height; ++y) {
            NSColor *newColor = NSReadPixel(NSMakePoint(x, y));
            if (close) {
                if ((InRange([color hueComponent],
                              [newColor hueComponent], .036)) &&
                    (InRange([color brightnessComponent],
                              [newColor brightnessComponent], .05))) {
                    found++;
                }
            } else {
                if (([color redComponent] == [newColor redComponent]) &&
                    ([color greenComponent] == [newColor greenComponent]) &&
                    ([color blueComponent] == [newColor blueComponent])) {
                    found++;
                }
            }
        }
    }

    // Unlock the image focus and return the result of our search
    [m_image unlockFocus];
    return (ignore ? (found>1) : (found>0));
}
```

After looking at this code, can you think of any ways to optimize it? Try swapping the loops so that you walk **Y** before you walk **X**. Does this speed up your program? With a small watch box, you probably won't notice any difference, but on a large map with a large watch box, you just might. You might also consider walking the image data directly, avoiding the use of `NSReadPixel` altogether. These graphics optimizations

are beyond the scope of this book but something you might consider should you write more graphics-oriented software in the future.

Note

To use `NSReadPixel` to access the individual pixels of an image, you must wrap your pixel accesses with calls to `NSImage -lockFocus` and `NSImage -unlockFocus`. This is similar to having to use `SetPort` in previous Macintosh operating systems.

`RadarView` has it pretty easy: keep track of an image and a watch box and then use that information to match the colors in the image. You could easily make `RadarView` call out to a more specialized image analysis class to perform the color matching. It could even send the image to another computer for analysis if it really needed to crunch numbers.

Application Preferences

The last thing I want to talk about in `RadarWatcherX` is the application-level preferences. `RadarWatcherX` documents have plenty of settings that are document specific. Radar map URLs, colors to look for, reload intervals, and so on. However, other settings are more global in nature that `RadarWatcherX` keeps track of. These include whether to open a new window when the application starts up and how to alert the user when a matching color is found within a watch box.

Yet Another Nib and Some Code

As you saw in [Figure 3.2](#), there is a third nib file in the `RadarWatcherX` project. `Preferences.nib` contains the `NSWindow` shown in [Figure 3.11](#). `PreferenceController`, which is a subclass of `NSWindowController`, is the File's Owner. `PreferenceController` is defined as shown in [Listing 3.14](#). Note the extern `NSStrings` used to store the preferences and the three `IBOutlets` and `IBActions` for the controls.

Listing 3.14 PreferenceController.h

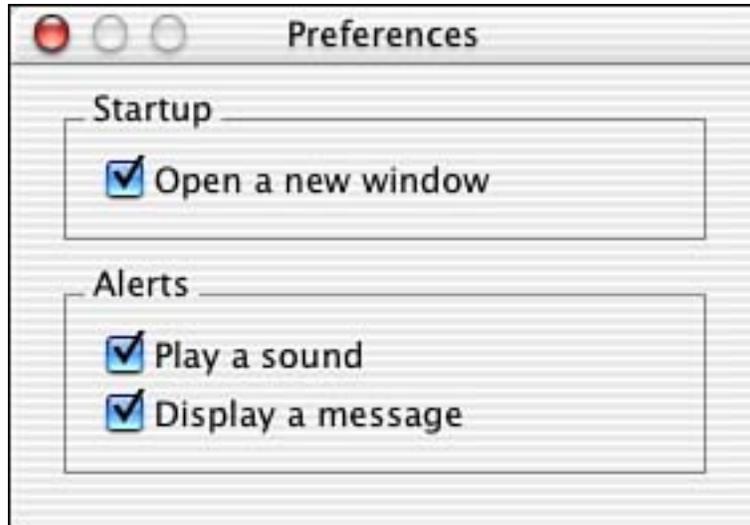
```
#import <AppKit/AppKit.h>

extern NSString *pkeyInitialized;
extern NSString *pkeyOpenNewWindow;
extern NSString *pkeyPlaySound;
extern NSString *pkeyShowAlert;

@interface PreferenceController : NSWindowController {
    IBOutlet NSButton *m_newWindowButton;
    IBOutlet NSButton *m_playSoundButton;
    IBOutlet NSButton *m_showAlertButton;
}
- (IBAction) changeNewWindowButton: (id) sender;
- (IBAction) changePlaySoundButton: (id) sender;
- (IBAction) changeShowAlertButton: (id) sender;

@end
```

Figure 3.11. RadarWatcherX application-level preferences.



Before we look at the code that makes the `PreferenceController` work, let's see how it is accessed from our `AppController`, shown in [Listing 3.15](#). First, when the `AppController` is initialized, it takes a moment to default our preferences should that be needed. By loading the `+standardUserDefaults` dictionary from `NSUserDefaults` and seeing if they have already been initialized, we can set up our default preference values easily. Remember the `Info.plist` entry for `CFBundleIdentifier`? This is the name of the file used to store our preferences in `~/Library/Preferences/`. The `com.triplesoft.radarwatcherx.plist` file is created at this point by `NSUserDefaults`.

Listing 3.15 `AppController +initialize`

```
+ (void) initialize
{
    // Create the user defaults dictionary
    NSUserDefaults *defaults;
    defaults = [NSUserDefaults standardUserDefaults];

    if ([defaults integerForKey:pkeyInitialized] == 0) {
        // Save our defaults if not already initialized
        [defaults setObject:[NSNumber numberWithInt:1]
            forKey:pkeyInitialized];
        [defaults setObject:[NSNumber numberWithInt:NSOnState]
            forKey:pkeyOpenNewWindow];
        [defaults setObject:[NSNumber numberWithInt:NSOnState]
            forKey:pkeyPlaySound];
        [defaults setObject:[NSNumber numberWithInt:NSOnState]
            forKey:pkeyShowAlert];
    }
}
```

Note

Note that the `+initialize` method is executed when the application's icon is still bouncing in the dock. If you needed to do any time-consuming processing at application startup, you

might consider doing it in the `-applicationDidFinishLaunching:` delegate notification instead. This notification is sent once the icon has stopped bouncing and can make your application feel snappier.

Now, any portion of our application can check a preference value with a simple call to `NSUserDefaults` as demonstrated when the application asks its delegate, `AppDelegate`, if it should open an untitled file when it starts up.

Listing 3.16 `AppDelegate -applicationShouldOpenUntitledFile:`

```
- (BOOL) applicationShouldOpenUntitledFile: (NSApplication *) sender
{
    return ([[NSUserDefaults standardUserDefaults]
            integerForKey:pkeyOpenNewWindow] == NSOnState);
}
```

Note

`-applicationShouldOpenUntitledFile:`, shown in [Listing 3.16](#), is an example of a delegate method. These methods are implemented by the delegate of a class and are called from the instance of the class. For example, the delegate of `NSApplication`, in this case our `AppDelegate`, can implement the `-applicationShouldTerminateAfterLastWindowClosed:` method to let the application know if it should terminate after the last window is closed. Remember, we receive these delegate methods because we registered `AppDelegate` as an application delegate in the nib file.

Delegate methods are a great way to add custom behaviors to a structured framework. You can see all the available delegate methods in the Cocoa documentation. You use Interface Builder to set up a delegate relationship just as you connect actions and outlets. Simply connect the delegate outlet of the class in question to the instance that is the delegate. Once connected, implement the delegate methods you are interested in and respond accordingly. Remember, not all objects allow delegates, and those that do differ from class to class. That is, `NSView` does not use delegates at all, and the delegates implemented in `NSApp` are different from those in `NSWindow`.

When the user selects Preferences from the application menu, the `AppDelegate -showPreferencePanel:` method is executed, as shown in [Listing 3.17](#). At this time, if the `preferenceController` instance variable has not been allocated, it is. The `-showWindow:` method of `NSWindowController` is then called to display the preferences. Later on, when the application quits and the `AppDelegate` is deallocated, it also releases the `preferenceController`. The `preferenceController` sticks around for the entire life of the application once it has been created.

Listing 3.17 `AppDelegate -showPreferencePanel: /-dealloc`

```
- (IBAction) showPreferencePanel: (id) sender
{
```

```

// Create the PreferenceController if it doesn't already exist
if (!preferenceController)
    preferenceController = [[PreferenceController alloc] init];

// Display it
[preferenceController showWindow:self];
}

- (void)dealloc
{
    // We are done with the PreferenceController, release it
    [preferenceController release];
    [super dealloc];
}

```

PreferenceController Code

The `PreferenceController` code itself, shown in [Listing 3.18](#), is rather straightforward with just a few tricks. Upon being initialized by `AppController`, the `PreferenceController` must call its super - `initWithWindowNibName:`, passing the proper nib name—in this case, `Preferences`.

Listing 3.18 PreferenceController.m

```

#import "PreferenceController.h"

NSString *pkeyInitialized = @"pkeyInitialized";
NSString *pkeyOpenNewWindow = @"pkeyOpenNewWindow";
NSString *pkeyPlaySound = @"pkeyPlaySound";
NSString *pkeyShowAlert = @"pkeyShowAlert";

@implementation PreferenceController

- (id)init
{
    // Load the Preferences.nib file
    self = [super initWithWindowNibName:@"Preferences"];
    return self;
}

- (void)windowDidLoad
{
   NSUserDefaults *defaults;

    // Load our default values and set the preference controls accordingly
    defaults = [NSUserDefaults standardUserDefaults];
    [_newWindowButton setState:[defaults integerForKey:pkeyOpenNewWindow]];
    [_playSoundButton setState:[defaults integerForKey:pkeyPlaySound]];
    [_showAlertButton setState:[defaults integerForKey:pkeyShowAlert]];
}

- (IBAction)changeNewWindowButton:(id)sender
{
    // Save back the new value of this control to the defaults
    [[NSUserDefaults standardUserDefaults] setInteger:[sender state]

```

```

        forKey: pkeyOpenNewWindow];
    }

- (IBAction) changePlaySoundButton: (id) sender
{
    // Save back the new value of this control to the defaults
    [[NSUserDefaults standardUserDefaults] setInteger: [sender state]
     forKey: pkeyPlaySound];
}

- (IBAction) changeShowAlertButton: (id) sender
{
    // Save back the new value of this control to the defaults
    [[NSUserDefaults standardUserDefaults] setInteger: [sender state]
     forKey: pkeyShowAlert];
}

@end

```

Then, when `-windowDidLoad` is called after the nib loads our window, we can access the `+standardUserDefaults` to set the initial state of the controls in the Preferences window.

Last, when each button is clicked and the state changes, we write the current value back to the `+standardUserDefaults`. This means that as soon as a check box is clicked, the value is live. If another portion of the application attempted to access the value, it would be in sync with the current state of the check box. Note, however, that the file may not be written to disk immediately. `NSUserDefaults` does not provide concurrency between two processes.

That's about all there is to it!

[\[Team LiB \]](#)

Try This

Here are some ideas to expand RadarWatcherX; try these on your own.

Add slider controls to the document window to adjust the sensitivity of the `closeColors` option. Adding one for hue and one for brightness sensitivity should do the trick and make the application more controllable, depending on the maps the user chooses.

Add the ability to make more than one watch box in a document. Multiple watch boxes with their own color selections could really expand the usefulness of the application.

Add the ability to email a user as an alert option. You can use the `-deliverMessage: subject: to:` method of the `NSMailDeliver` class in Apple's Message framework located in `/System/Library/Frameworks`. Although no formal documentation exists for this framework, extensive notes are in the header file. While you're there, look at the header files from other frameworks you don't recognize. You can learn a lot by poking around (read-only) in the depths of your system's directory structure!

Conclusion

You learned a lot in this chapter. The combination of looking through the source code while reading the enhanced explanations here will help you to get a grasp on designing and implementing a document-based Cocoa application of your own.

If you are confused, reread the chapter one section at a time and concentrate on the relationships between the source code in Project Builder and the user interface items in Interface Builder. Cocoa allows you to almost transparently link these items together. Through the magic of the Cocoa framework, they all become connected at runtime.

Also, make sure to learn to use the debugger in Project Builder. It may be confusing and overwhelm you at first, but, trust me, if you learn a few commands (that is, 'po,' also known as print object), you will be able to understand more about what is happening in your application. Set breakpoints on all your functions, especially delegate methods, to see when they are called and what parameters are being passed. Type '**help**' in the debugger for more information.

With a little commitment and some hard work, it will all start to come together for you and you will be poised to build the next killer app!

Chapter 4. Cocoa Plug-ins

"...the documentation doesn't specify (there is no documentation)..."

—nibs

Where would web be without plug-ins? Plug-ins are also known as plug ins, PlugIns, or whatever you want to call them. A plug-in is usually a file or bundle that is created and compiled separately from a host application, that follows a specific API (application programming interface) or protocol, and that is used by the host application to extend its functionality in some way.

Adobe Photoshop uses plug-ins to allow third-party developers to extend the functionality of its image editing capabilities. By publishing an API and making it available to the public, Adobe has essentially "opened up" Photoshop to developers everywhere. These developers can create plug-ins that work directly within the Photoshop application environment. In addition, they do not require access to the original Photoshop source code, undoubtedly valuable to Adobe!

Music software also uses plug-ins to allow editing of digital audio. Plug-ins in the audio world include those that create reverb, echo, and other effects. The same concept applies; third-party developers download an SDK (software development kit) that provides all the information necessary to communicate with the host application via a plug-in API. Most of the time, a plug-in accepts some type of data (audio or image data, for example), processes it, and returns the processed version to the host. The host then displays the alterations to the user. The user need not be concerned that the host application was written by one company and the plug-in by another—it all just works!

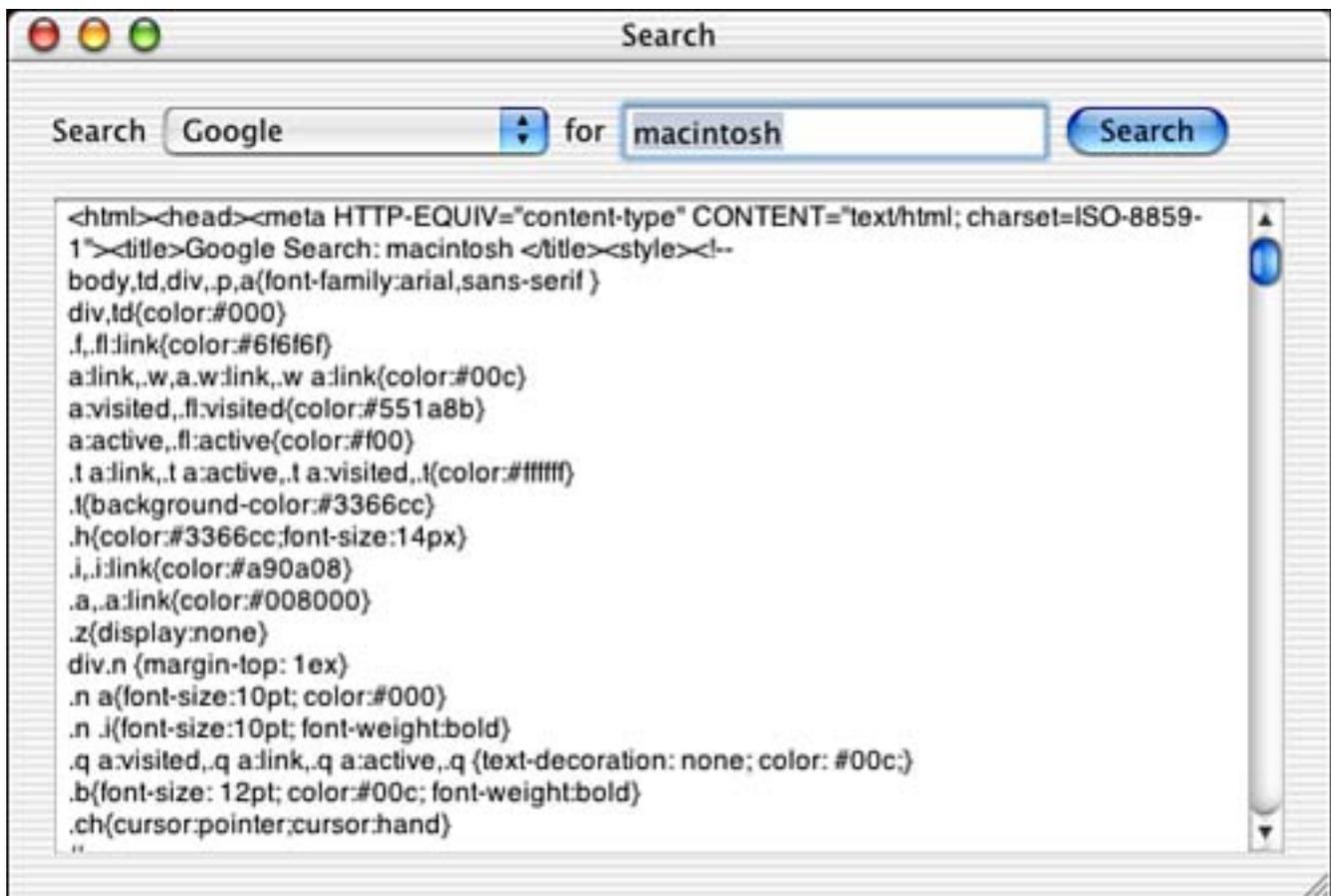
Within OS X are numerous ways to provide plug-in capabilities to an application. This chapter discusses two methods that work well within Cocoa applications. [Chapter 5](#), "Carbon Plug-ins," discusses writing plug-ins using a Carbon-based API for use within Carbon or Cocoa applications.

The Results

There are two results in this chapter.

First, I will discuss a plug-in architecture that uses what I refer to as "dumb" plug-ins. These plug-ins are nothing more than text-based plist configuration files that are loaded in by the application, parsed, and used to provide options to the user. In our case, we provide a search window that allows you to choose from a variety of Internet search engines to query for a search term. The raw HTML results are then displayed in a scrolling `NSTextView`. The contents of the search pop-up button are loaded from the available plug-in plist files. This makes the application easily extensible without altering the underlying source code.

Figure 4.1. PlistPlugin search engine application window.

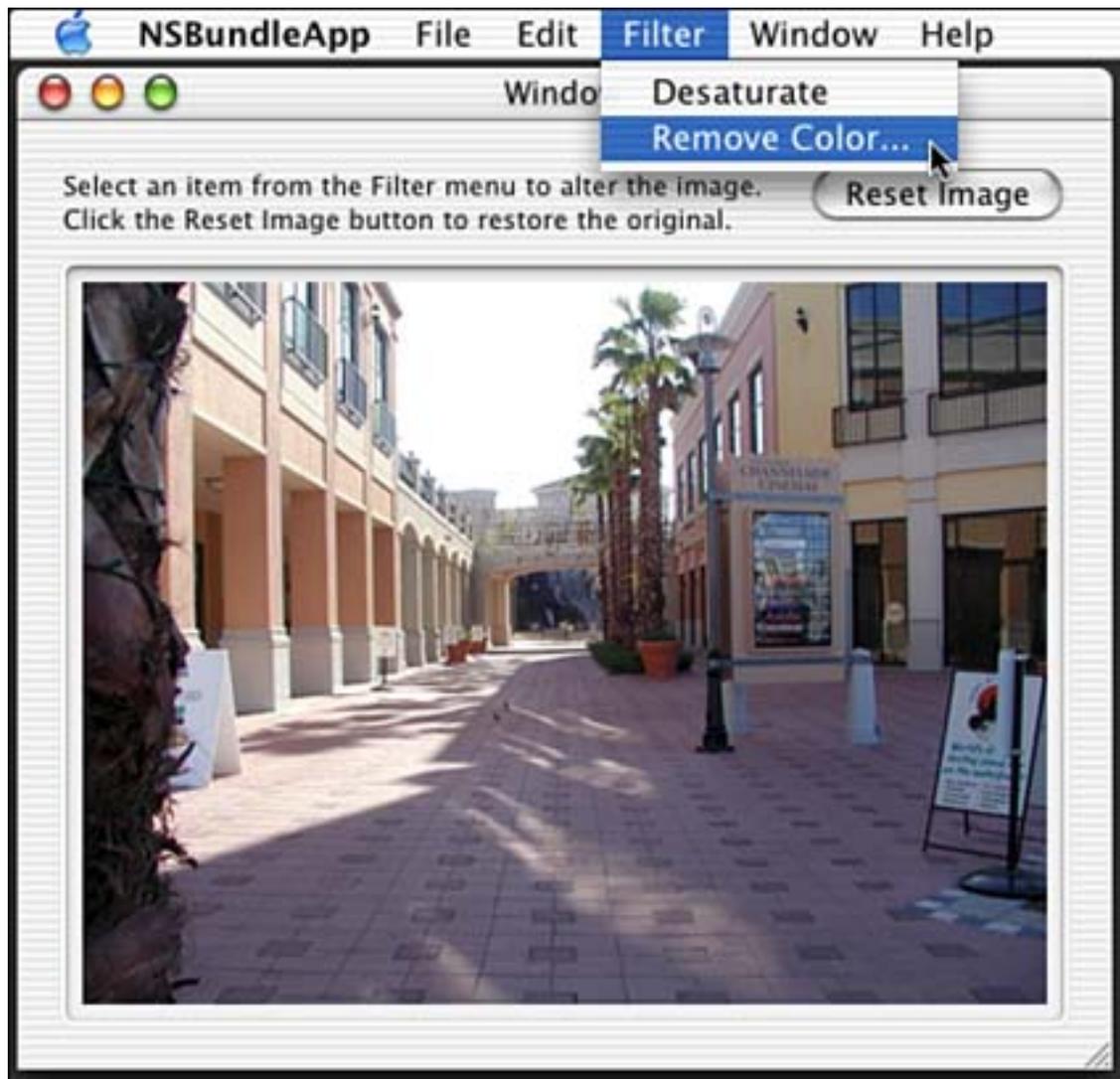


Second, I will discuss a plug-in architecture based on the `NSBundle` class. The host application loads plug-ins that, in and of themselves, are bundles and call methods within them to provide image-editing capabilities. Each plug-in is listed in and can be selected from the Filter menu. The application and the plug-in bundles adhere to a specific, albeit simplistic, protocol to allow quick and easy communication between the two. The plug-ins are passed the image, alter it, and pass it back to the host application for display—just like Photoshop, but maybe not as advanced.

Both of the projects in this chapter are simple Cocoa applications used to feature the architecture implementations over anything else. In a real application, you may choose to implement a document-based

approach depending on your needs.

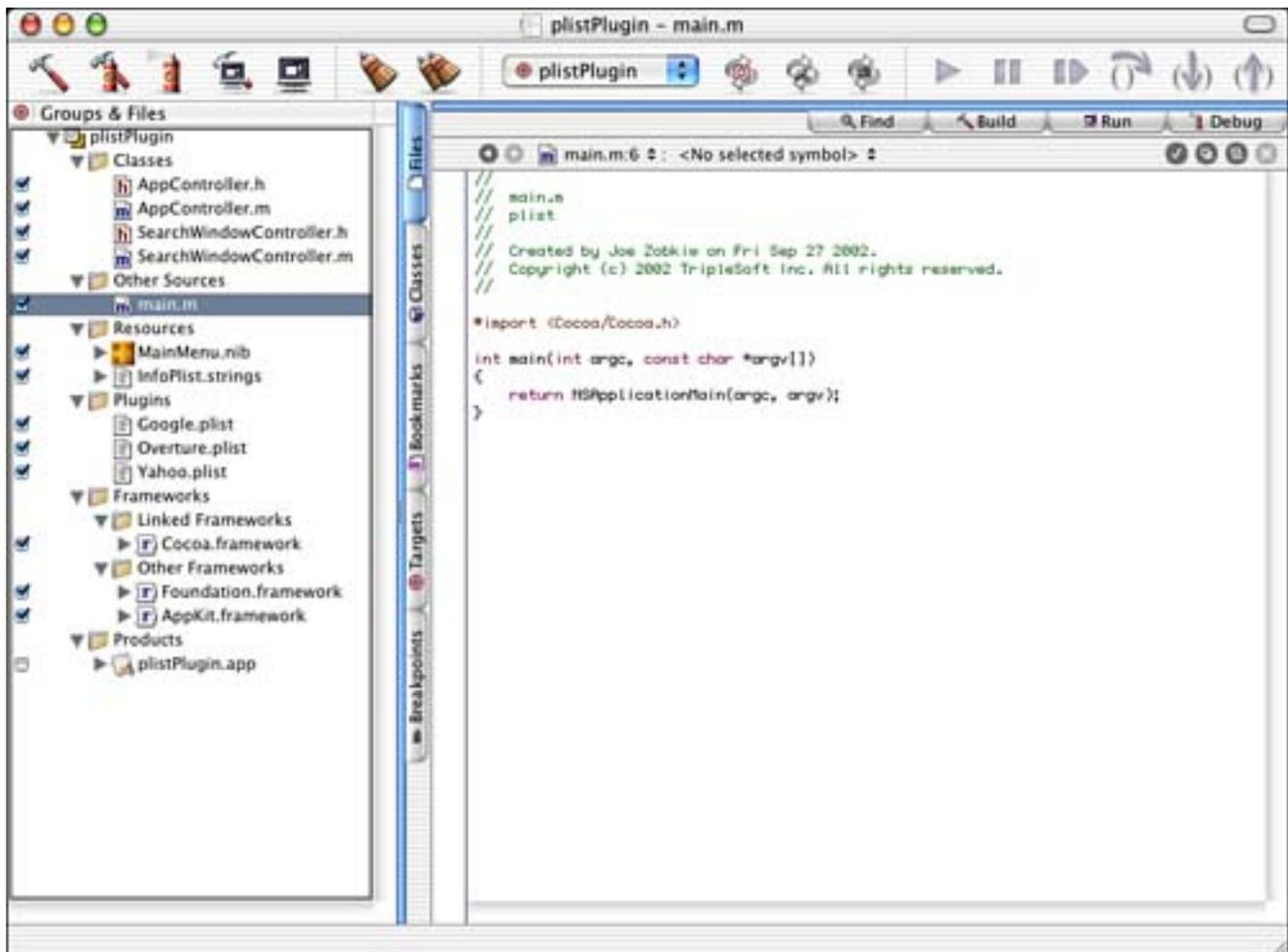
Figure 4.2. NSBundle image editing application window.



The First Project: plistPlugin

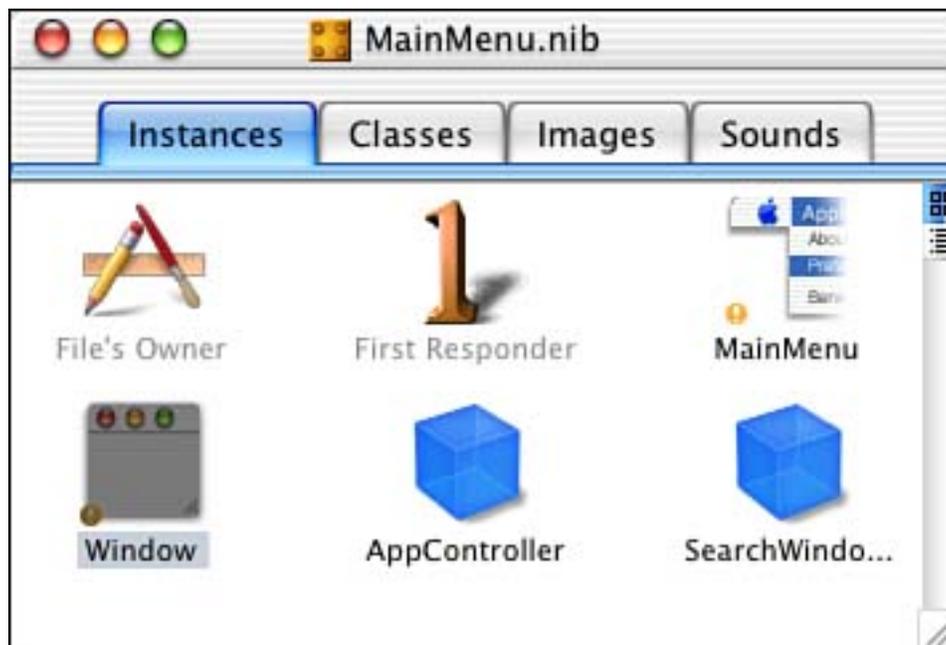
As you can see in [Figure 4.3](#), this project is straightforward. We have an `AppController`, which we only use to make a delegate of the `NSApplication` so that we can implement the `-applicationShouldTerminateAfterLastWindowClosed:` method. By implementing this method, you can quit the application simply by closing the main window. This convenience is used in most of our example applications.

Figure 4.3. The plistPlugin project in Project Builder.



`SearchWindowController` is where all the action is. This is the class that not only controls the window (responding to the Search button), but also loads the plug-ins. [Figure 4.4](#) illustrates how our nib file is arranged. Both `AppController` and `SearchWindowController` are instantiated, and connections between the Window and `SearchWindowController` are as you would expect for a window with a few outlets and an action.

Figure 4.4. The plistPlugin MainMenu.nib in Interface Builder.



Before we look at the details of how the plug-ins are loaded and called, let's talk about the plug-ins themselves for a moment.

The Plug-ins: What

You can see that we created a Plug-ins group in the project file. In that group, we added the three plug-ins that we created for this project—one for each of three different search engines that the program supports (see [Figure 4.5](#)). These plug-ins are nothing more than XML format text files ending in .plist.

Figure 4.5. The Google.plist plug-in.



The files contain two keys that are used by the application. The **SearchEngineName** key value contains the name of the search engine as it should be displayed in the pop-up button list. The **SearchEngineURL** key value contains the URL that is used to access the search engine. The search term entered by the user is appended to the end of the **SearchEngineURL** key value. In the case of Google, the **SearchEngineURL** key value starts out as

http://www.google.com/search?q=

Once the host application appends the search term to it, it will look like

<http://www.google.com/search?q=macintosh>

If you were to type this URL in to an Internet browser, you will see that it is all Google needs to perform a search.

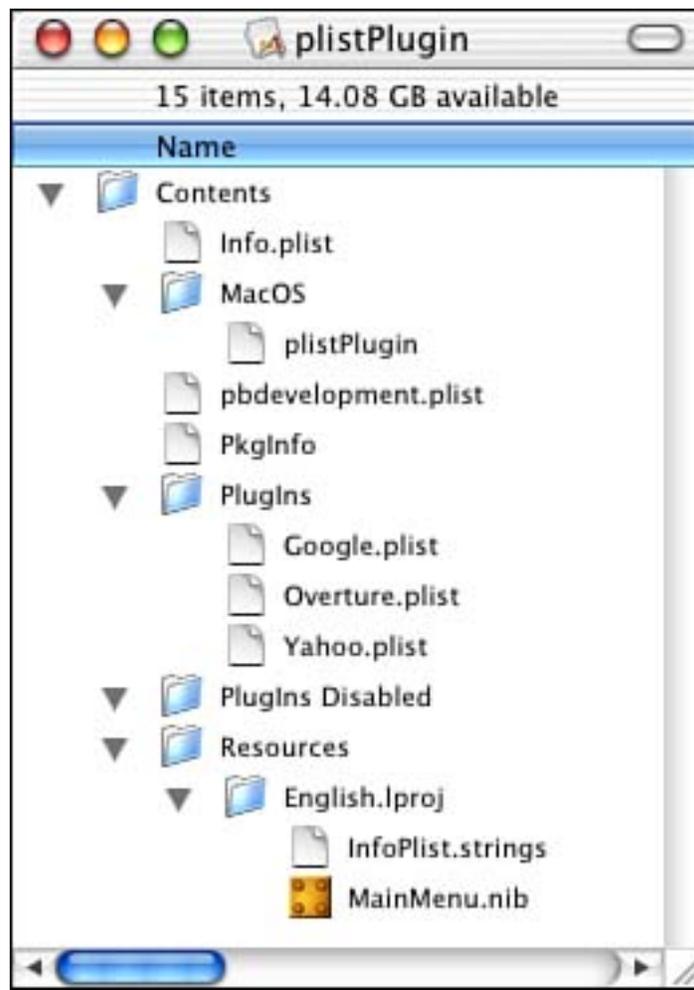
Note

Google runs on a Web server. When we access the preceding URL, we are contacting Google's Web server and asking it to perform a search of its database and return the results. You can easily write a Web server script using PHP, or some other scripting language, and access it remotely from your application using this same method. You need not limit this technique to search engines alone. This is how some applications implement the "Internet Version Check" that we will demonstrate in a later chapter.

The Plug-ins: Where

But where are these plug-ins? The plug-ins themselves live in the PlugIns directory inside the host applications bundle. See [Figure 4.6](#) for the hierarchical layout of the compiled application. Note that the plug-ins are all gathered quietly together within the application itself. You can see this as well by control clicking on the plistPlugin application and selecting Show Package Contents from the pop-up menu.

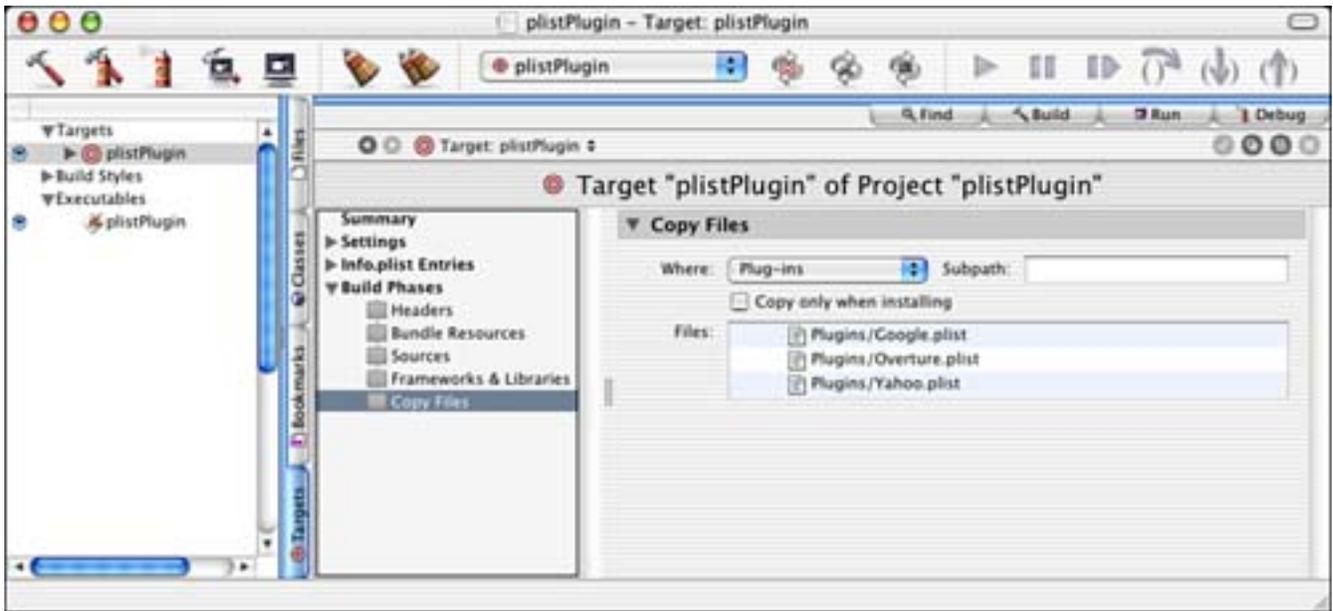
Figure 4.6. The plistPlugin application package contents.



The Plug-ins: How

How did the plug-ins get there? [Figure 4.7](#) shows the Copy Files settings of the plistPlugin Target in the project. By adding a Copy Files Build Phase, we are able to choose which files get copied where within the application bundle. In our case, we chose to have the three search engine plug-ins copied to the Plug-ins directory. (See what I mean about the many capitalization versions of plug-in—even within Apple's own software.) If you are not familiar with Project Builder in this capacity, be sure to consult the Project Builder documentation.

Figure 4.7. The plistPlugin Target Copy Files settings.

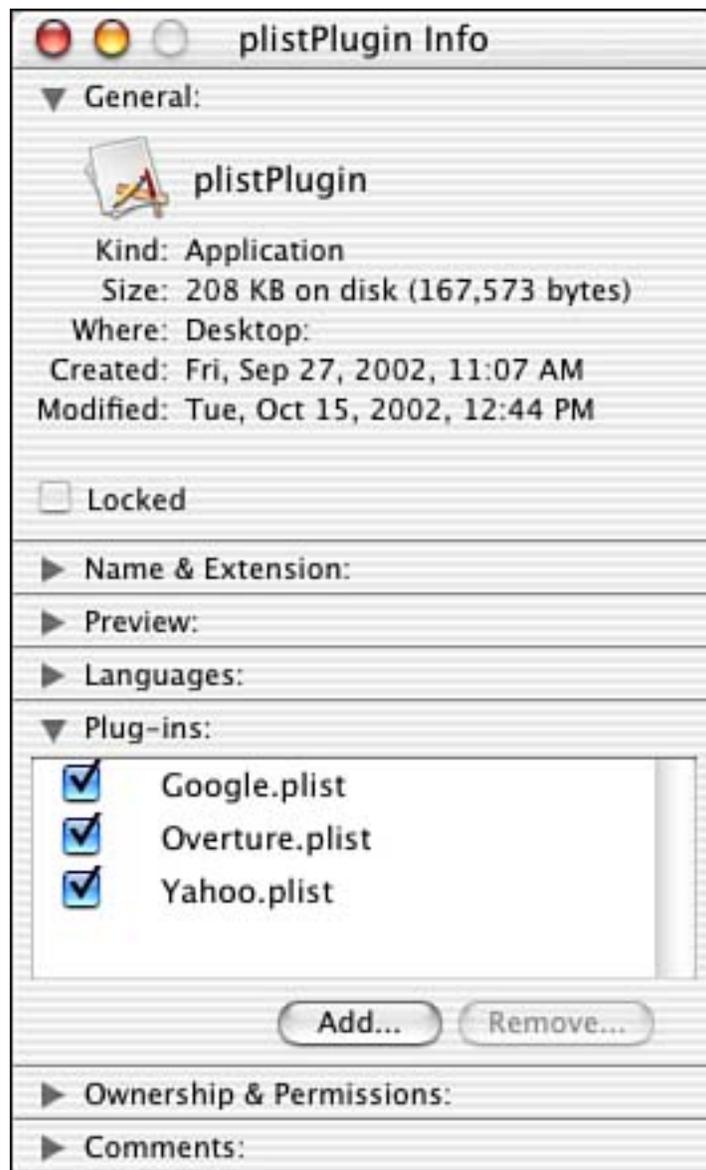


Note

You can actually store the plug-ins in many different locations. They do not have to be within the application that uses them. In fact, many third-party plug-ins are more likely to be installed in the user's `~/Library/Application Support/` directory. You might consider placing the plug-ins in your application's Resources directory if they need to be localized (that is, `Contents/Resources/English.lproj/Google.plist`).

An interesting twist on using plug-ins stored in the PlugIns directory within the application is that you can enable and disable them easily from within the Get Info window of the application itself. [Figure 4.8](#) shows the plug-ins that are within the `plistPlugin` application. By simply turning off the ones we don't want and restarting the application, they will be disabled. That's a nice feature!

Figure 4.8. The `plistPlugin` application Info window.



SearchWindowController

Now we'll move on to the source code to see how this all fits together. As mentioned previously, the `SearchWindowController` is where all the action is. It has two methods that handle 99% of its work. - `awakeFromNib` loads the plug-ins and is called automatically by the Cocoa framework once the main window in the nib file is loaded. When the user clicks the Search button, the - `doSearch:` method appends the search term to the search URL, initiates the search, and displays the results. Let's look at these methods.

Listing 4.1 - `awakeFromNib`

```
// Called to initialize our fields if we need to -
// this is where we will load our plugins
- (void)awakeFromNib
{
    NSString* folderPath;

    // Remove all items from the popup button
    [m_searchEnginePopupButton removeAllItems];

    // Locate the plugins directory within our app package
    folderPath = [[NSBundle mainBundle] builtInPlugInsPath];
    if (folderPath) {
        // Enumerate through each plugin
        NSEnumerator* enumerator = [[NSBundle
            pathsForResourceOfType:@"plist"
            inDirectory:folderPath] objectEnumerator];
        NSString* pluginPath;
        while ((pluginPath = [enumerator nextObject])) {
            // Load the plist into a dictionary for each plugin
            NSDictionary* dictionary =
                [NSDictionary dictionaryWithContentsOfFile:pluginPath];
            if (dictionary) {
                // Read the items in the plist
                NSString* searchEngineURL =
                    [dictionary objectForKey:@"SearchEngineURL"];
                NSString* searchEngineName =
                    [dictionary objectForKey:@"SearchEngineName"];
                if (searchEngineName && searchEngineURL) {
                    // Add the string the array
                    [m_searchEngineArray addObject:searchEngineURL];
                    // and add a menu item (menu items and
                    // array items will have the same index,
                    // which makes it easy to match them up
                    // when the user presses the search button)
                    [m_searchEnginePopupButton addItemWithTitle:searchEngineName];
                }
            }
        }
    }
}
```

Don't let [Listing 4.1](#) scare you; once you take it apart, it isn't half as bad as it looks. After clearing out the pop-up button, we use `NSBundle` to return the `-builtInPlugInsPath`—that is, the path to the Plugins directory within our application bundle. We then enumerate through the Plugins directory for all files ending in `.plist`, which is how our plug-ins are named. Then we load the contents of each plug-in file into a temporary dictionary. Have I mentioned how great property list files are for storing `NSDictionary` data? After we have the file contents in the dictionary, we can easily extract the `SearchEngineURL` and the `SearchEngineName` key values. Assuming that we get them both (that is, the file is not corrupt), we add the URL to an array that is an instance variable of our `SearchWindowController` and add the name to our pop-up button list. The index of the URL in the array will always be in sync with the index of the name in the pop-up button, which will come in handy in the next method!

Note

If you chose to store your plug-ins in the Resource directory as mentioned earlier, you would change the enumeration logic in [Listing 4.1](#) to look in specific lproj-like directories for the plugins (that is, `Contents/Resources/English.lproj/Google.plist`).

[Listing 4.2](#) contains the action that is called when the user presses the Search button. As noted previously, the array of URLs and the pop-up button items have the same array indices. Therefore, we can simply pull the URL from the array using the index of the currently selected item in the pop-up button list. We then use one of the Core Foundation routines to encode the search term into the proper format for URLs. This will turn a space into a `%20`, for example. Once encoded, we concatenate the strings to create the complete search URL. We then use `NSURL` to load the contents of the URL, and if results are returned, we display them in the scrolling text field.

Listing 4.2 - doSearch:

```
// This action is called when the user clicks the Search button
- (IBAction)doSearch: (id) sender
{
    NSString *results;

    // Since the menu items and array items both
    // have the same index, we can easily load the URL
    // based on the menu item that is selected
    NSString *searchEngineURL = [m_searchEngineArray
        objectAtIndex: [m_searchEnginePopUpButton indexOfSelectedItem]];

    // We must then encode the string that the user
    // typed in (ie: replacing spaces with %20)
    NSString *encodedSearchText =
        (NSString*) CFURLCreateStringByAddingPercentEscapes(NULL,
        (CFStringRef) [m_searchTextField stringValue],
        NULL, NULL, kCFStringEncodingUTF8);

    // Once encoded, we concatenate the two strings, the URL and the search text
    NSString *completeSearchURL =
        [NSString stringWithFormat:@"%%%@", searchEngineURL,
        encodedSearchText];
```

```

// Begin user feedback
[m_progressIndicator startAnimation:self];
[m_searchButton setEnabled:NO];

// We then attempt to load the URL and save the results in a string
results = [NSString stringWithContentsOfURL:
    [NSURL URLWithString:completeSearchURL]];
if (results) {
    // If we have results we display them in the text view
    NSRange theFullRange = NSMakeRange(0,
        [[m_resultsTextView string] length]);
    [m_resultsTextView replaceCharactersInRange:theFullRange
        withString:results];
} else {
    NSRunAlertPanel(@"Error",
        @"NSString stringWithContentsOfURL returned nil",
        @"OK", @"", @"");
}

// End user feedback
[m_searchButton setEnabled:YES];
[m_progressIndicator stopAnimation:self];
}

```

As you can see, there really isn't much to this. It's a simple way to implement a plug-in architecture that not only allows your application to be extended easily, but also gives the user some control over which plug-ins are enabled and disabled. You can even publish an API that would allow non-programmers to extend your application using this mechanism. Remember, no programming is required for the plug-in creator in this example.

Note

Many extremely useful Core Foundation routines can be used directly with Cocoa objects. All you have to do is typecast your Cocoa `NSString`, for example, to the proper type as done previously. When you can substitute a Cocoa object for a Core Foundation data type with little more than a typecast, this is called "toll-free bridging." This also works the other way around; some Core Foundation data types can be "toll-free bridged" to Cocoa objects. Note that not all objects are bridgeable. `NSBundle` and `CFBundle`, for example, are not bridgeable. When in doubt, check the Core Foundation documentation. `CFURL.h` and `CFString.h` are two header files to begin your explorations.

The Second Project: MyNSBP_App and MyNSBP_Plugin

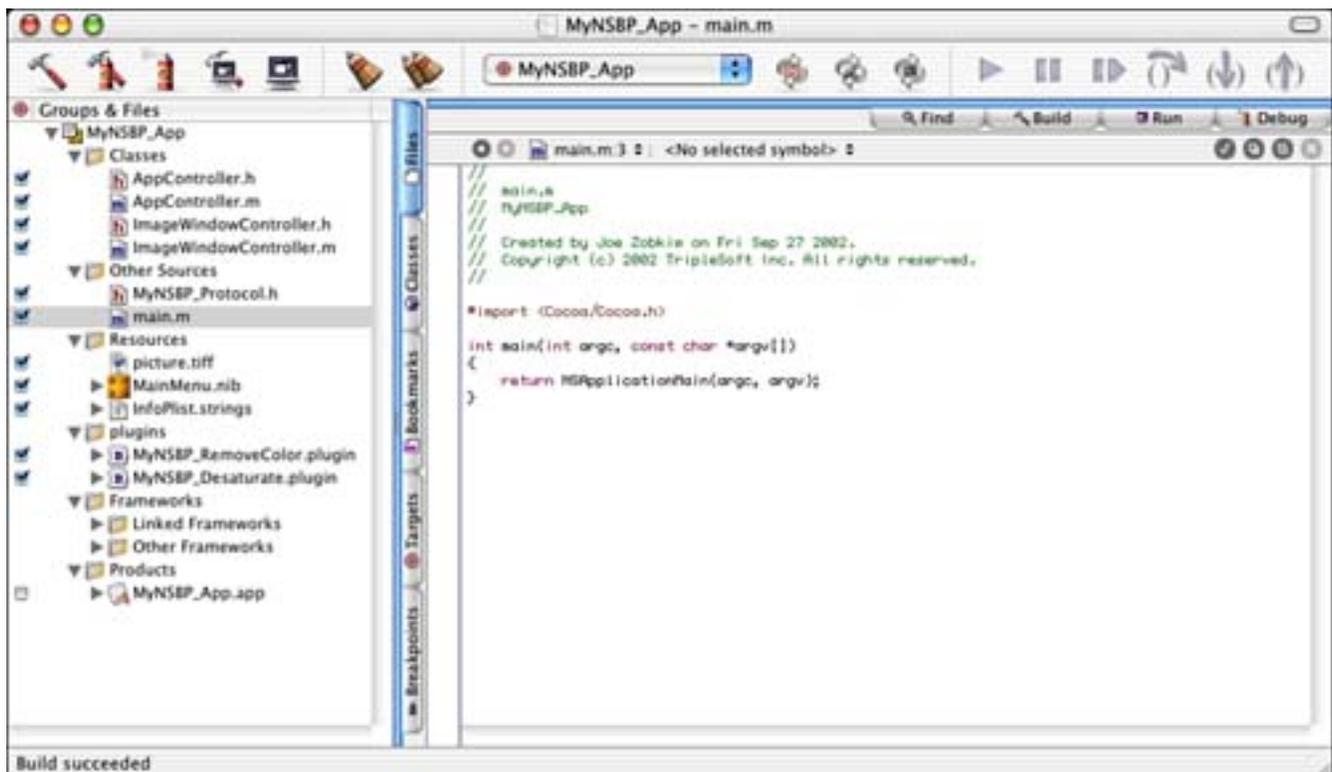
The second Cocoa plug-in project actually consists of multiple project files. There is a host application project file and then a plug-in project file. In this case, the application will display the image and manage the plug-ins. The plug-in will alter the image by removing one of the primary colors, red, green or blue, as chosen by the user.

Let's look at the two project files in general first, and then we will delve deeper into the source code of each. You will see many similarities between the previous plistPlugin project and this project.

The Host Application: Overview

Once again in this project, seen in [Figure 4.9](#), we use an `AppController` strictly for the `-applicationShouldTerminateAfterLastWindowClosed:` method—nothing else to say there. The `ImageWindowController`, an override of `NSObject`, manages the main application window and handles menu selections from the Filter menu. `MyNSBP_Protocol.h` houses the protocol that all plug-in filters follow in order to be used by this application.

Figure 4.9. The MyNSBP_App project in Project Builder.



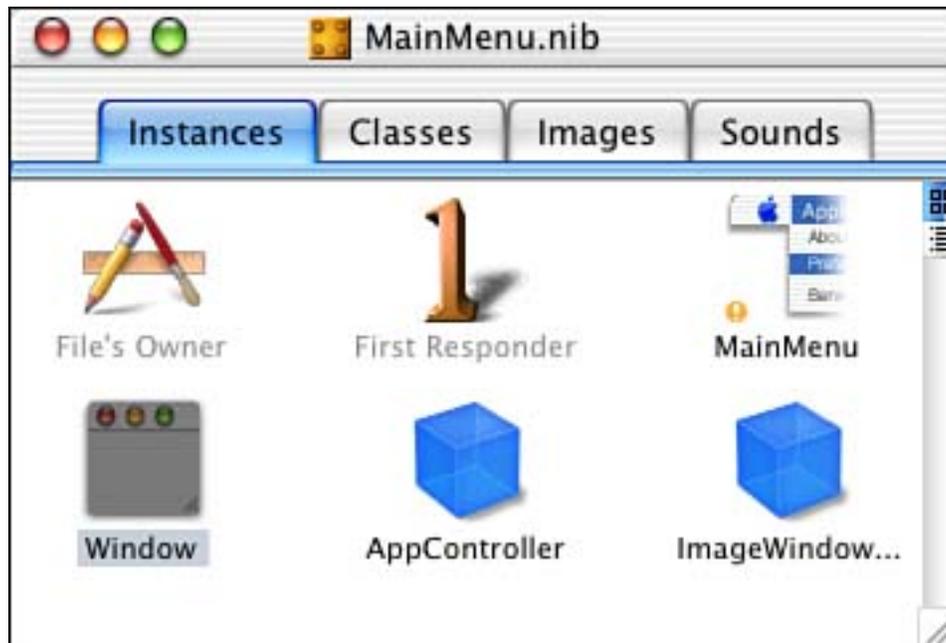
Note

Want another option? It seems that there are always options when programming for Mac OS X. Instead of using a protocol, we could have chosen to implement a public class interface that our third-party developers would override to create their custom filters. This class would be

the link between the application and the plug-in—much like our protocol is in this example. Both mechanisms are perfectly acceptable and are many times a matter of personal taste.

On the resource side of things, the TIFF file is used as our default image in the window—a colorful photo of Tampa Bay, Florida that was taken by esteemed photographer Ed McNair. `MainMenu.nib`, as seen in [Figure 4.10](#), contains the window resource as well as instantiations of the `AppController` and the `ImageWindowController`—standard stuff for a simple Cocoa application. `InfoPlist.strings` contains what you would expect as well—our copyright and version information.

Figure 4.10. The MyNSBP_App MainMenu.nib in Interface Builder.



You will also notice that we have created a plug-ins group containing two plug-ins that are generated from other projects—one of which will be discussed in a moment; the other is a bonus, and not discussed here. Further, if you were to look in the Target information for the `MyNSBP_App`, you will see that we set up another Copy Files Build Phase to copy the built plug-ins into the `PlugIns` directory within our application's bundle. This is the same type of setup we had in the `plistPlugin` example; the only difference is in this case we are copying entire bundles into the `PlugIns` directory and not just `.plist` files. Shortly, you will see how our application loads these enhanced plug-ins as compared to loading the `.plist` files.

The Plugin: Overview

The plug-in project, seen in [Figure 4.11](#), will look subtly different from the others you've seen. The most important thing to note is that this project does not build an application. Instead, the plug-in created is a bundle. The project was started by using the Project Builder New Project Assistant, seen in [Figure 4.12](#), and creating a Cocoa Bundle.

Figure 4.11. The MyNSBP_RemoveColor project in Project Builder.

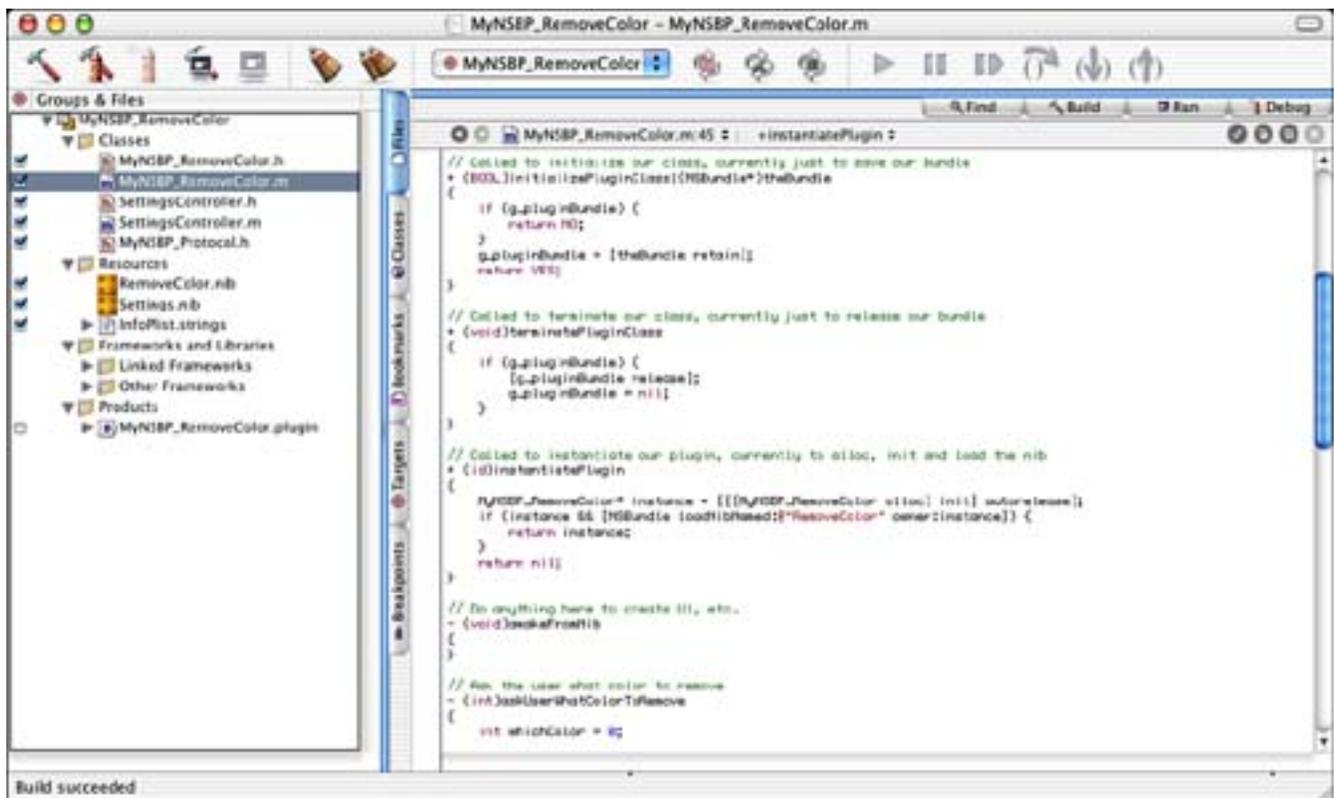
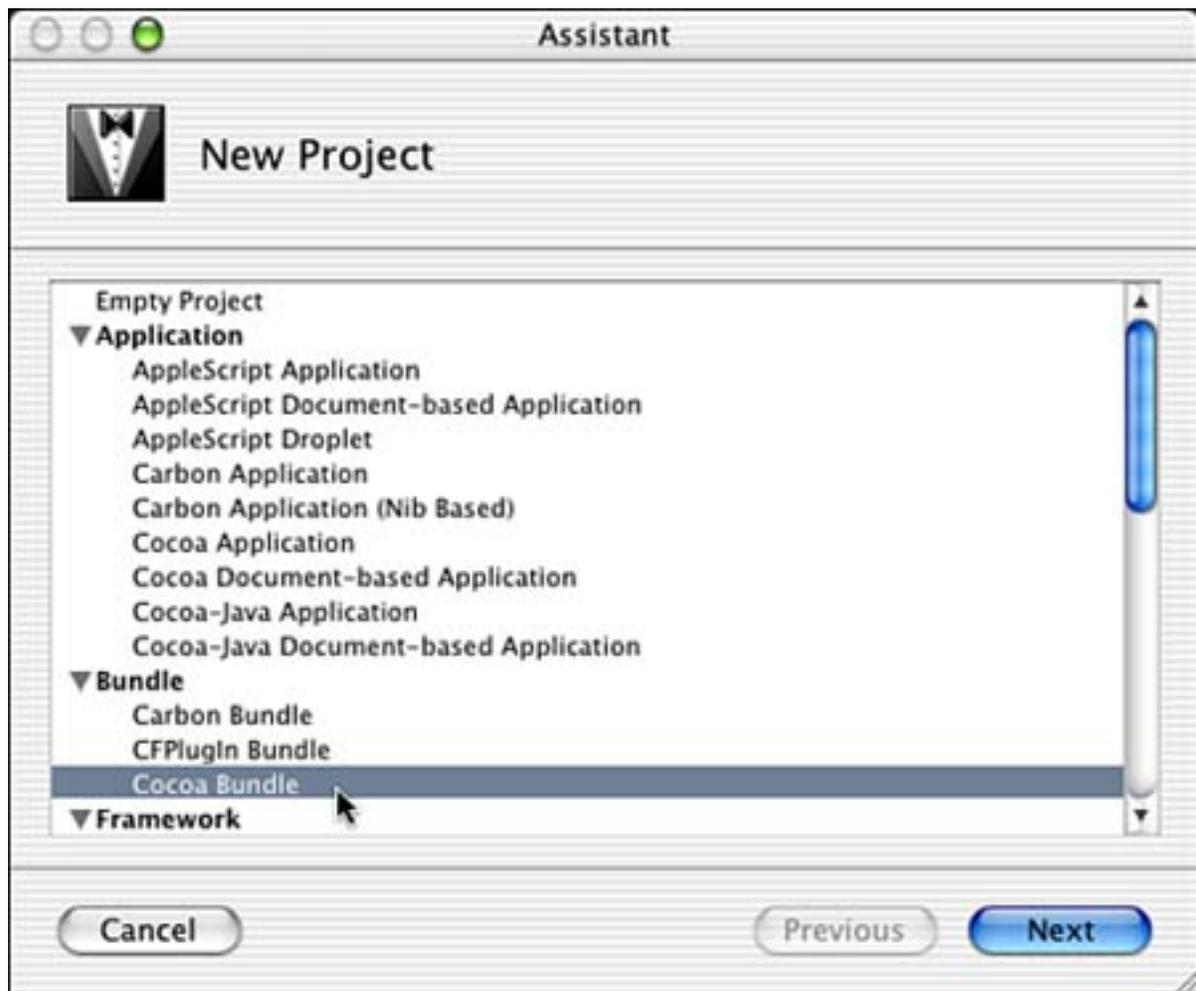
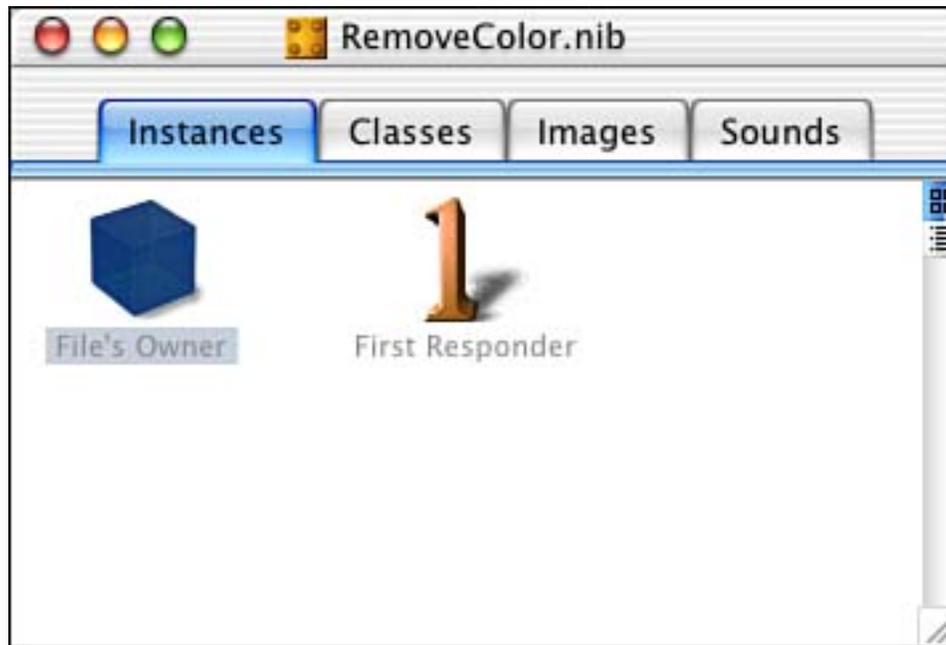


Figure 4.12. The Project Builder New Project Assistant.



`MyNSBP_RemoveColor` is the guts of our plug-in. The `MyNSBP_RemoveColor` class is a subclass of `NSObject` that adheres to the `MyNSBP_Protocol` protocol. This is also the principal class of our project. Notice that the plug-in project includes `MyNSBP_Protocol.h` just as the host application does. The associated `RemoveColor.nib` file, as shown in [Figure 4.13](#), is essentially empty, containing only the default File's Owner and First Responder. The File's Owner is set to the principal class `MyNSBP_RemoveColor`. You could leave this empty .nib file out if you so desire; I left it in the project in case we needed to add something later during development.

Figure 4.13. RemoveColor.nib in Interface Builder.

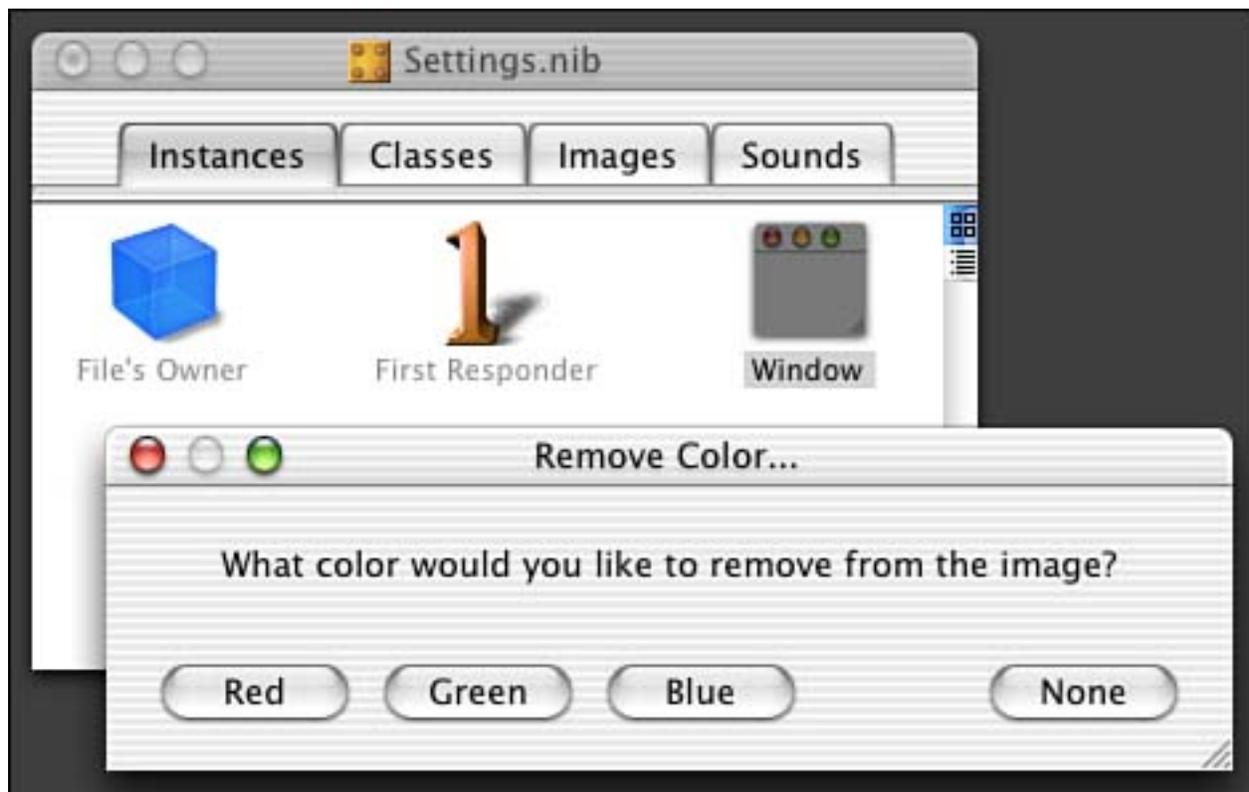


Note

A principal class is essentially a custom, named main entry point into a bundle. You can typically access all the functionality in a bundle through its principal class. `NSApplication` is usually the principal class when you are creating an application. However, in this case, the principal class is that of our plug-in, `MyNSBP_RemoveColor`. Properly defining the principal class is important, as you will see later on when our application has to call our plug-in based on this value. The principal class typically acts as a traffic cop because it controls all other classes in a bundle. It also manages interactions between internal classes and those outside the bundle.

Getting back to classes, the `SettingsController` class is an override of `NSWindowController` and is simply used to manage the modal window that is displayed when you choose this particular filter as shown in [Figure 4.14](#). The associated `Settings.nib` file is used in conjunction with the `SettingsController` class mentioned previously, which happens to be its File's Owner. Note also that the window is connected to the outlet of the File's Owner—this is key.

Figure 4.14. Settings.nib in Interface Builder.



Let's look at the application and plug-in code in more detail.

The Host Application: Detail

As mentioned previously, the `ImageWindowController` is the meat of the host application project, shown in [Figure 4.15](#). It has only a few instance variables that are nothing more than outlets to the `NSMenuItem` (Filter menu) and the `NSImageView`. It also maintains two `NSMutableArray`s—one for a list of plug-in classes and the other for a list of plug-in instances. These arrays are allocated when the object is initialized and deallocated when it is destroyed. The arrays are used when actually calling one of the plug-ins. Let's look at the code used to load the plug-ins first in [Listing 4.3](#).

Listing 4.3 `ImageWindowController`'s `awakeFromNib`

```
// Called to initialize our fields if we need to -
// this is where we will load our plugins
- (void)awakeFromNib
{
    NSMenu      *filterMenu = [m_filterMenuItem submenu];
    NSString    *folderPath;

    // Locate the plugins directory within our app package
    folderPath = [[NSBundle mainBundle] builtInPluginsPath];
    if (folderPath) {

        // Enumerate through each plugin
        NSEnumerator *enumerator =
            [[NSBundle pathsForResource:@"plugin"
            inDirectory:folderPath] objectEnumerator];
        NSString    *pluginPath;
```

```

int                count = 0;

while ((pluginPath = [enumerator nextObject])) {

    // Get the bundle that goes along with the plugin
    NSBundle* pluginBundle = [NSBundle bundleWithPath: pluginPath];
    if (pluginBundle) {

        // Load the plist into a dictionary
        NSDictionary* dictionary = [pluginBundle infoDictionary];

        // Pull useful information from the plist dictionary
        NSString* pluginName = [dictionary
            objectForKey:@"NSPrincipalClass"];
        NSString* menuItemName = [dictionary
            objectForKey:@"NSMenuItemName"];

        if (pluginName && menuItemName) {

            // See if the class is already loaded, if not, load
            Class pluginClass = NSClassFromString(pluginName);

            if (!pluginClass) {
                NSObject<MyNSBP_Protocol>* thePlugin;

                // The Principal Class of the Bundle is the plugin class
                pluginClass = [pluginBundle principalClass];

                // Make sure it conforms to our protocol and attempt
                // to initialize and instantiate it
                if ([pluginClass
                    conformsToProtocol:@protocol(MyNSBP_Protocol)] &&
                    [pluginClass isKindOfClass:[NSObject class]] &&
                    [pluginClass initializePluginClass:pluginBundle] &&
                    (thePlugin = [pluginClass instantiatePlugin])) {

                    // Add a menu item for this plugin
                    NSMenuItem *item;
                    item = [[NSMenuItem alloc] init];
                    [item setTitle:menuItemName];
                    [item setTag:count++];
                    [item setTarget:self];
                    [item setAction:@selector(doFilter:)];
                    [filterMenu addItem:item];
                    [item release];

                    // Add the class to our array
                    [m_pluginClasses addObject:pluginClass];

                    // Add the instance to our array
                    [m_pluginInstances addObject:thePlugin];
                }
            }
        }
    }
}

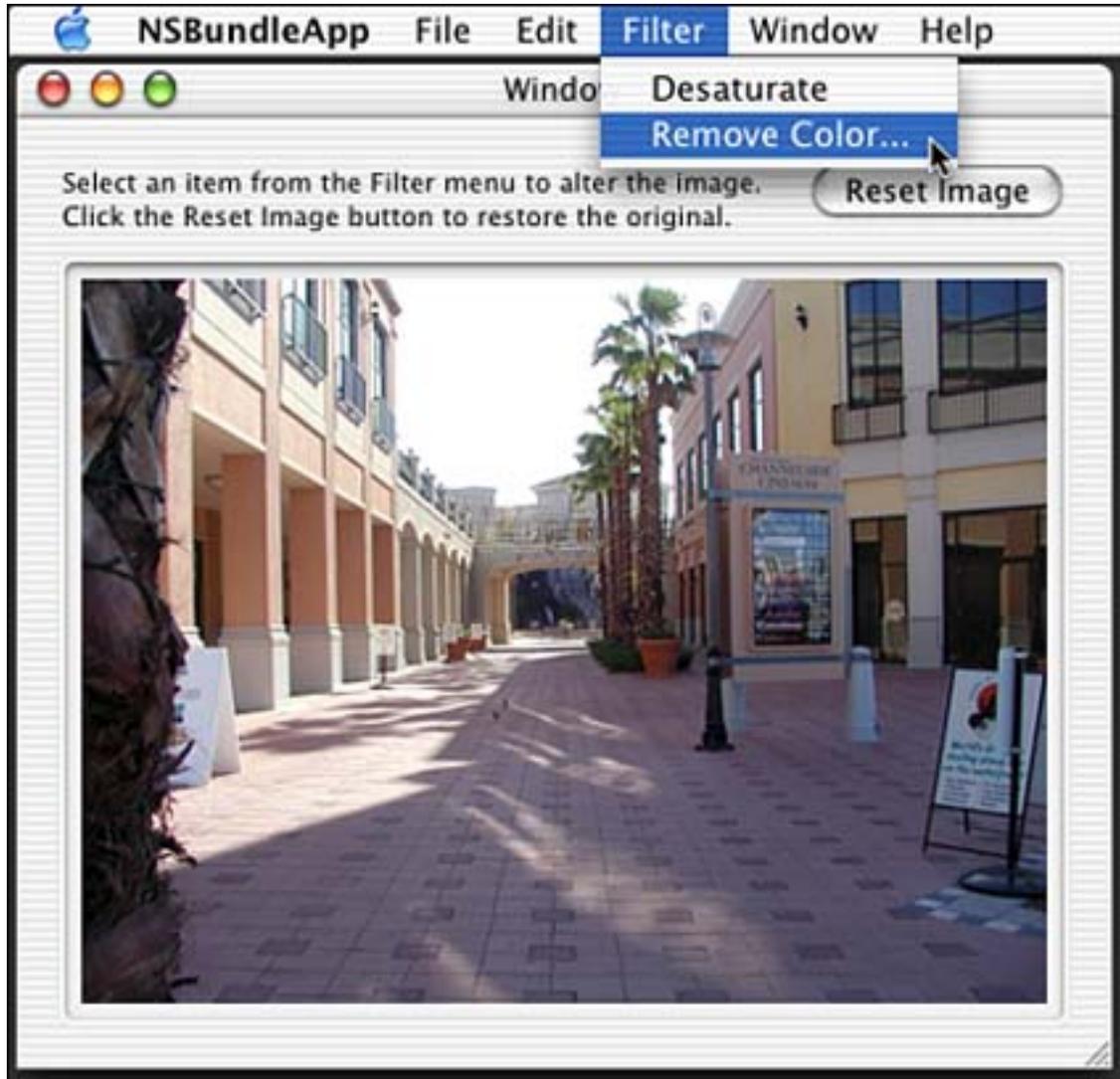
```

```

}
// Update our self image
[self doResetImage: self];
}

```

Figure 4.15. NSBundle image editing application window.



As you know, `-awakeFromNib` is called once our nib file has been completely loaded. We are guaranteed that all of our outlets are connected at this point. Therefore, this is the perfect method to initialize our user interface. In our case, we load our plug-ins and subsequently build the contents of the Filter menu.

Note

Where's a better place to load? Although we are loading our plug-ins in `-awakeFromNib` in this example, it might be better for your application to load them in `NSApplicationDelegate's -applicationDidFinishLaunching:` method. The reason is that during `-awakeFromNib`, your application's icon is still bouncing in the dock. By loading in `-applicationDidFinishLaunching:`, you have already surpassed the bouncing icon and your application load time will seem snappier to the user. This will become very apparent if you have many plug-ins to load at startup. You might also consider deferring the initialization and

instantiation of each plug-in until they are actually used—yet another optimization!

Also note that `-awakeFromNib` has a dark side. If you have more than one controller object in your nib file, you generally can't say which one will receive `-awakeFromNib` first. This can become a problem if one controller attempts to access the other for whatever reason and it is yet to completely exist!

The first thing we do is access the submenu of the Filter menu item so that we can add more items to it later. Then we access the main application bundle and request the `-builtInPlugInsPath`. Next, we enumerate the `PlugIns` directory for all files ending in `.plugin`. Remember that in the `plistPlugin` project, we enumerated for `.plist` files—same concept here.

For each plug-in that we find, we access its bundle using `NSBundle`'s `-bundleWithPath:` method passing the path to the plug-in. This gives us an `NSBundle` that we can then work with to dissect the plug-in and access the code and resources that we are interested in. Once we load the `info.plist` data into a dictionary, using the `-infoDictionary` method of `NSBundle`, we can begin to pull the key values from it. Once again, this is very similar in concept to `plistPlugin`.

The main key values we are interested in are the principal class, represented by `NSPrincipalClass`, and the menu item name, represented by `NSMenuItemName`. You will see how these are configured when I discuss the plug-in project shortly. For now, know that they exist; this will become clear once we look at the plug-in code in detail. Given the principal class name, `MyNSBP_RemoveColor`, we can access the class object itself for the plug-in by calling `NSClassFromString`. If this function returns a nil class, we know that the class has yet to be loaded from the bundle and we must load it.

To load the class, we declare an `NSObject` variable that conforms to the `MyNSBP_Protocol`—that is, a local variable that will represent our plug-in. We then call `NSBundle`'s `-principalClass` method, which returns the plug-in's principal class after ensuring that the code containing the definition itself is loaded. Simply put, the `-principalClass` method loads the code.

Note

After calling the `NSBundle -principalClass` method, which uses `-load` to load the code, the receiver posts an `NSBundleDidLoadNotification`. Using this mechanism, you can perform actions when the bundle has successfully loaded.

Now that we have the class, we want to verify that it is actually one of our plug-ins. The odds are that it will be. But you never know when a user might accidentally drag a plug-in in to our application bundle, so it doesn't hurt to check. We validate a few items as follows: Does the class conform to the `MyNSBP_Protocol`? Is it based on `NSObject`? If these check out, we call the `+initializePlugInClass:` class method of the plug-in to enable it to keep track of its own bundle and then the `+instantiatedPlugIn` class method to enable it to create itself. These will be discussed in more detail later.

We add the plug-in's name to the menu, setting the target to the `ImageWindowController` and the action to the `-doFilter:` method within it. You will also note that we set the tag value of the menu item to the index of the menu item. This allows us to figure out which plug-in item was selected from the menu and access the associated class and instance arrays, as in `plistPlugin`, by menu item index. You will see this in

detail in the `-doFilter:` method later. We then use `-addObject:` to append our plug-in class and instance to the two arrays. Once again, we depend on the synchronization of the items in the arrays and those in the Filter menu.

Note

You will notice that many classes throughout Cocoa have a `-tag` value associated with them. This tag can be set to any value you desire. If you are a Macintosh programmer from way back, you might have referred to a tag as a `refCon`.

Last, but not least, we call the `-doResetImage:` method of `ImageWindowController` (see [Listing 4.4](#)) to make our picture appear. Essentially, this method loads the TIFF from our bundle, creates an `NSImage` out of it, and sets it as the image of the `NSImageView` in our window. Note that we have to make a copy of the image otherwise the image itself is "edited" when we execute our filters on it. When this happens, there is no way to reset it without relaunching the application. That's the magic of caching!

Listing 4.4 -doResetImage:

```
// Called when the user pressed the Reset Image
// button, restores the image in the view
// We must make a copy otherwise the actual
// picture.tiff tends to get "edited"
- (IBAction)doResetImage: (id) sender
{
    NSImage *theImage = [[[NSImage imageNamed:@"picture.tiff"] copy] autorelease];
    [imageView setImage:theImage];
}
```

Note

If you were allowing the user to load an image from a file, instead of from your own bundle, you would use `-initWithContentsOfFile:` instead of `-initWithImageNamed:` to reset the image.

Okay, so now we have an image displayed in the window, a menu filled with filters, and nowhere to go. At this point, we expect the user to select an item from the Filter menu. Once they do this, the `-doFilter:` method is called in `ImageWindowController`. This method couldn't be simpler; let's take a look.

Listing 4.5 -doFilter:

```
// Called when one of the plugins is selected from the Filter menu
- (IBAction)doFilter: (id) sender
{
    NSObject<MyNSBP_Protocol>* thePlugin;

    // Get the proper plugin instance from the array based on tag value
    thePlugin = [m_pluginInstances objectAtIndex:[sender tag]];
}
```

```

if (thePlugin) {
    // Pass the image to the plugin, which will alter it and pass it back
    NSImage *theAlteredImage = [thePlugin doPlugin: [m_imageView image]];

    // Set the altered image to be the current image
    [m_imageView setImage: theAlteredImage];

    // Update the view
    [m_imageView setNeedsDisplay: YES];
}
}

```

Because each Filter menu item calls this method as its action, we need a way to figure out which plug-in was selected from the menu. Remember the `-tag` value? By retrieving the `-tag` (the index) of the sender (the menu item), we can tell which plug-in was selected. We then pull that index from the instances array, and, like magic, we have the plug-in instance.

Now that we have the plug-in instance, we call the `-doPlugin:` method within it, passing the image from the `NSImageView`. The plug-in will manipulate it and return it as `theAlteredImage`. We finish off by setting `theAlteredImage` as the image of the `NSImageView` and forcing it to update with `-setNeedsDisplay:`.

One last thing to note is that when the window is about to close, we have to release the memory occupied by our plug-in classes and instances. In [Listing 4.6](#), we simply enumerate the arrays and release the objects in them. The one exception is that we call the plug-in's `+terminatePluginClass` class method to balance the `+initializePluginClass` class method called during `-awakeFromNib` for the items in the classes array.

Listing 4.6 - windowWillClose:

```

// Called when the window is about to close
- (void)windowWillClose: (NSNotification*) notification
{
    Class                pluginClass;
    NSEnumerator         *enumerator;
    NSObject<MyNSBP_Protocol> *thePlugin;

    // Enumerate through the instances and release each
    enumerator = [m_pluginInstances objectEnumerator];
    while ((thePlugin = [enumerator nextObject])) {
        [thePlugin release];
        thePlugin = nil;
    }

    // Enumerate through the classes and terminate each
    enumerator = [m_pluginClasses objectEnumerator];
    while ((pluginClass = [enumerator nextObject])) {
        [pluginClass terminatePluginClass];
        pluginClass = nil;
    }
}

```

Remember, you've only seen one side of the architecture up to this point. Once you later see how the plug-in and protocol work, you might want to reread this section and then everything should become clear. Feel

free to take a break at this point if your mind is feeling numb. This is some complex stuff, especially if it is completely new to you. Take it slow, and don't be surprised if you have to review it a few times before it makes complete sense.

The Plug-in Protocol

Before diving into the plug-in itself, let's talk for a moment about the protocol that the plug-in conforms to. A protocol is a list of method declarations not directly associated with any particular class definition. Any class can adopt the protocol by implementing the methods and adding the protocol name to its interface definition. For example, the `MyNSBP_RemoveColor` class is a subclass of `NSObject` that implements the `MyNSBP_Protocol`, as you can see in [Listing 4.7](#).

Listing 4.7 `MyNSBP_RemoveColor` Interface Definition

```
@interface MyNSBP_RemoveColor : NSObject<MyNSBP_Protocol> {  
}  
- (int)askUserWhatColorToRemove;  
@end
```

So how is the protocol defined? [Listing 4.8](#) shows the `MyNSBP_Protocol` definition from `MyNSBP_Protocol`. Note that this protocol contains four methods—three class methods and one instance method. The class methods can be called without instantiating an object. The instance method requires a valid object before it can be called. This is standard Objective-C and shouldn't be any surprise.

Listing 4.8 `MyNSBP_Protocol` Definition

```
@protocol MyNSBP_Protocol  
  
+ (BOOL)initializePluginClass: (NSBundle*)theBundle;  
+ (void)terminatePluginClass;  
+ (id)initializePlugin;  
- (NSImage*)doPlugin: (NSImage*)theImage;  
  
@end
```

At this point, `MyNSBP_RemoveColor` need only implement the functions in the protocol. As you saw in the `MyNSBP_App` implementation, once you have a class, you can query it to see if it conforms to a protocol by using the `-conformsToProtocol:` method. The protocol defined here is known as a formal protocol. There are also informal protocols, but they do not allow the use of the `-conformsToProtocol:` mechanism, for example. Protocols are useful for declaring methods for other objects to implement, to hide a class definition yet provide an interface to an object, or to illustrate and provide for similarities between classes from different hierarchies.

Note

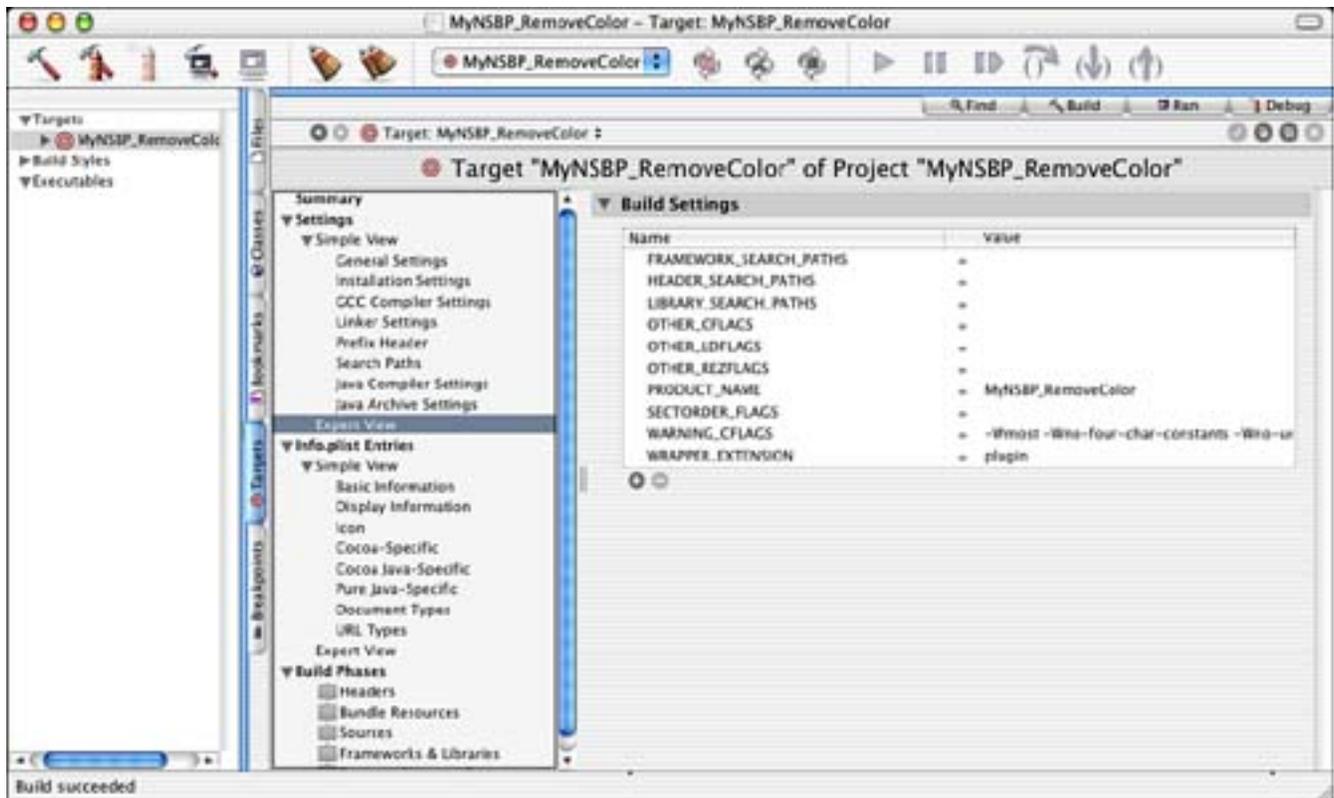
You can learn more than you ever thought you wanted to know about protocols in "The Objective-C Programming Language" publication from Apple. In fact, this is a great reference in any case to have on hand as you program in Objective-C. A PDF version of this document is available on the Apple Developer Web site at <http://www.apple.com/developer/>.

The Plug-in: Detail

You made it to the plug-in! We need to spend a few moments in the project settings before diving into the source code. Because this project is not standard to what we are used to seeing, there are a few things to point out that should be edited.

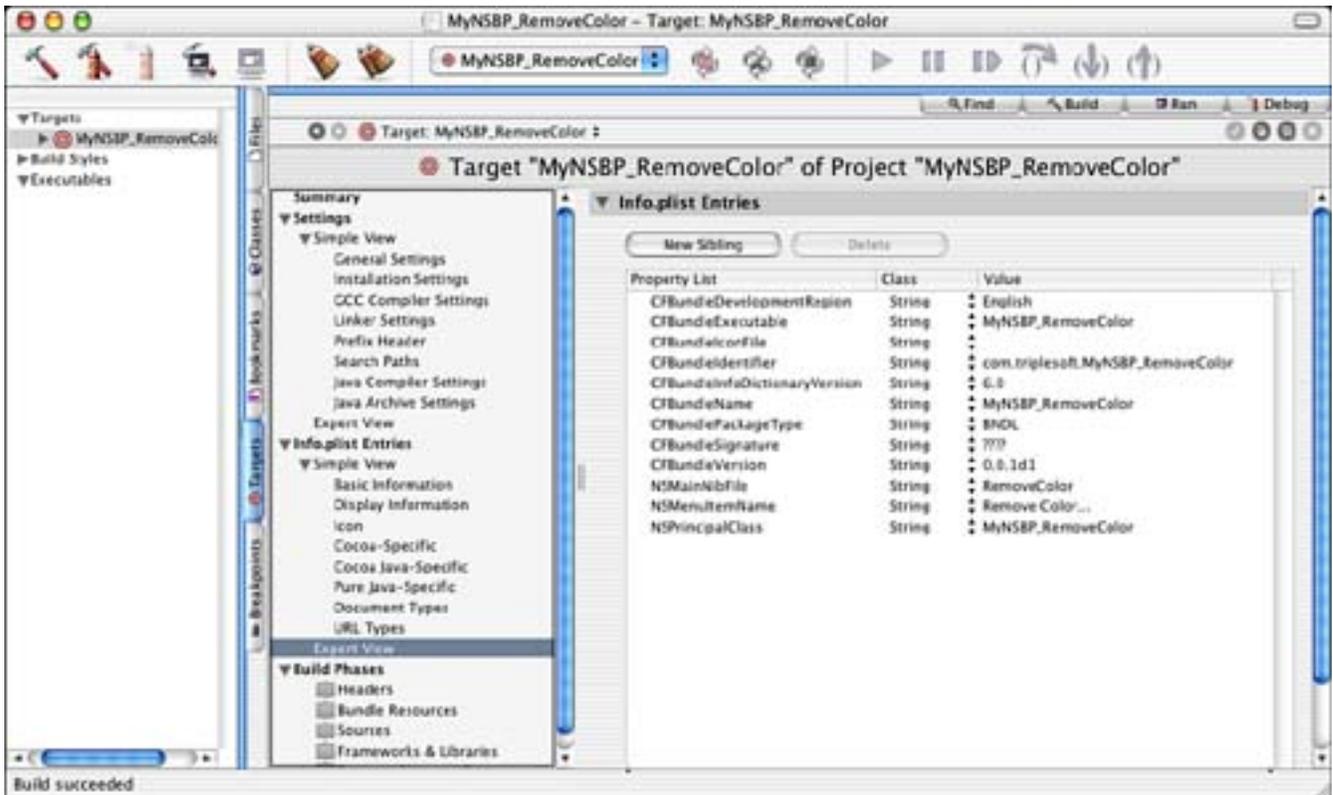
First in the Build Settings (see [Figure 4.16](#)), we need to change the `WRAPPER_EXTENSION` key to be `plugin` instead of the default `bundle`. This is the filename extension that will be used when our plug-in is built. In reality, it doesn't matter what we make this extension, but `plugin` is simply a standard. Because we are placing our final built plug-ins in the `PlugIns` directory, why not make them end in `.plugin`?

Figure 4.16. MyNSBP_RemoveColor build settings.



Second, in the `Info.plist` Entries (see [Figure 4.17](#)) we see the standard items that we have seen in the past. However, two in particular need to be pointed out. The `NSMenuItemName` key is set to `Remove Color`. This text is used in the Filter menu. The `NSPrincipalClass` is `MyNSBP_RemoveColor` for this project. This class implements the `MyNSBP_Protocol` protocol. You saw both of these values accessed in the `MyNSBP_App's -awakeFromNib` method; now you see where they were configured.

Figure 4.17. MyNSBP_RemoveColor Info.plist entries.



Let's move on to the source code!

Figures 4.18 and 4.19 show the before and after of calling the `MyNSBP_RemoveColor` to remove green from the image. Note that the Remove Color window is actually within the plug-in itself and called modally from it. You'll see how this all works in the next few pages.

Figure 4.18. About to remove green using `MyNSBP_RemoveColor`.

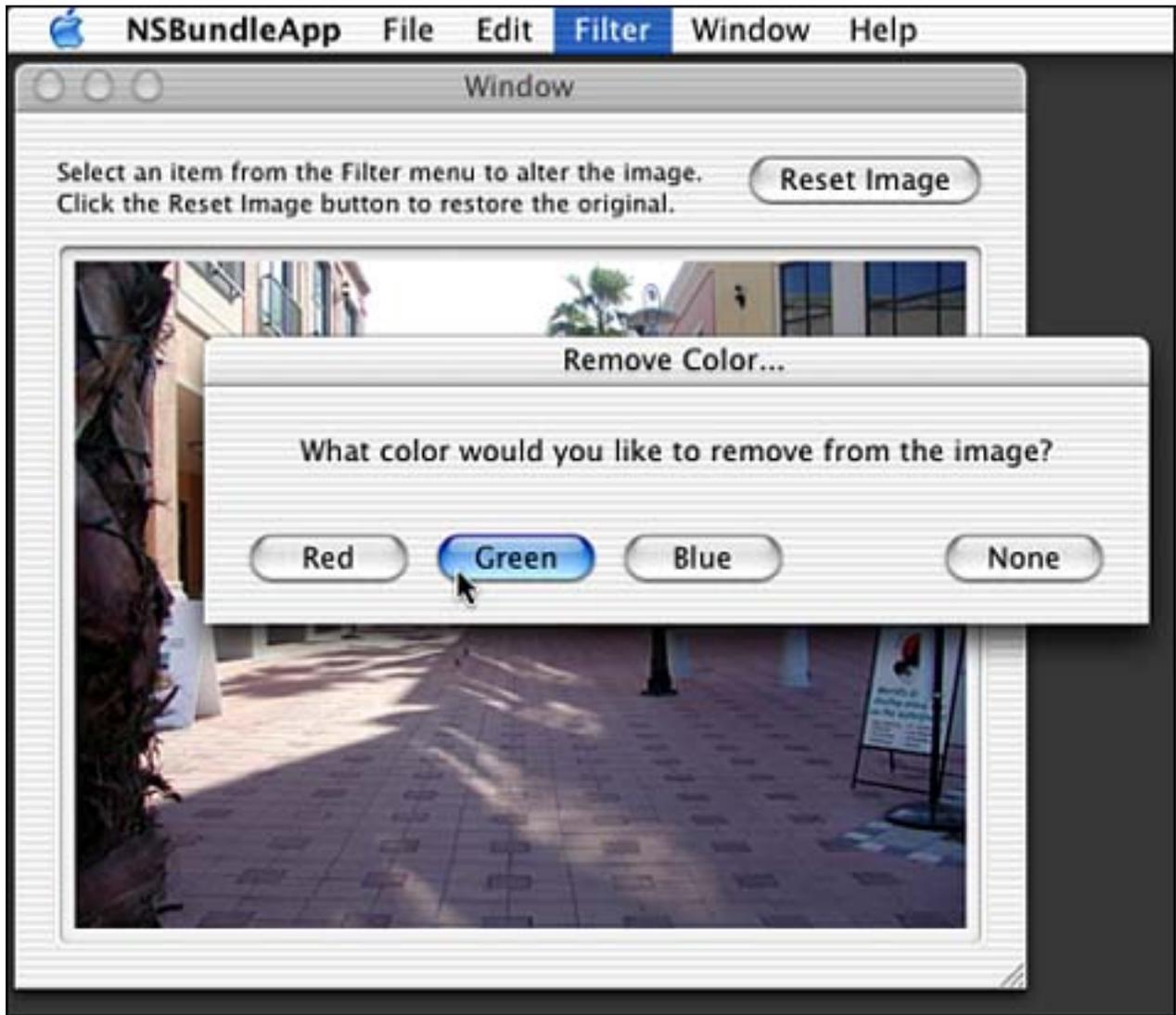
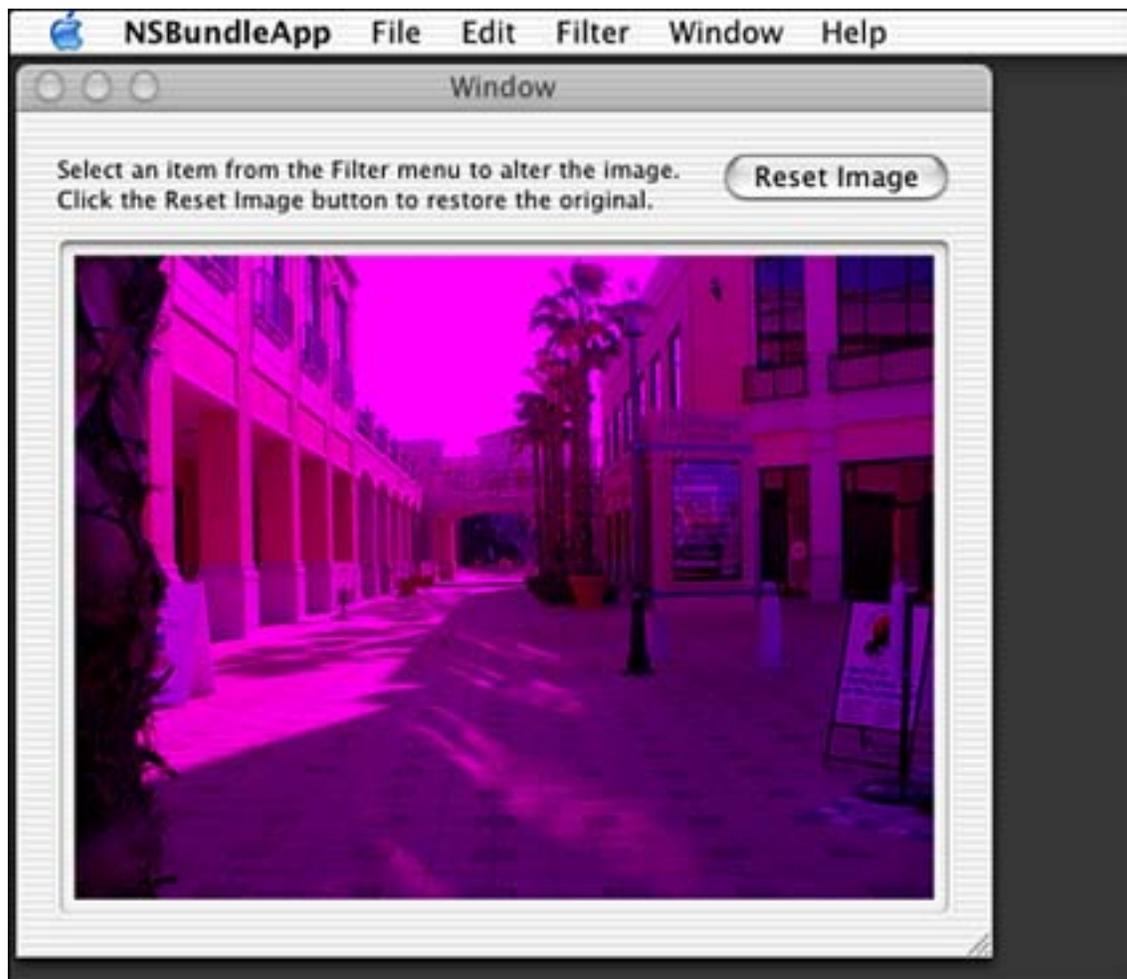


Figure 4.19. Having just removed green using MyNSBP_RemoveColor.



As you saw in the `MyNSBP_App -awakeFromNi b` method, the plug-in class is initialized with a call to the plug-in's `+i n i t i a l i z e P l u g i n C l a s s:` class method. The plug-in's bundle is passed to this method so that it can `-r e t a i n` it for later use. But this is not necessary and is only an example of doing so. The plug-in's `+t e r m i n a t e P l u g i n C l a s s` class method is called from `MyNSBP_App's -w i n d o w W i l l C l o s e:` method to allow it to clean up and, in our case, release its bundle before the plug-in itself is released by the host application. See [Listing 4.9](#) for the implementations of these methods in the plug-in.

Listing 4.9 `MyNSBP_RemoveCol or +i n i t i a l i z e P l u g i n C l a s s: / +t e r m i n a t e P l u g i n C l a s s`

```
// Called to initialize our class, currently just to save our bundle
+ (BOOL) i n i t i a l i z e P l u g i n C l a s s: (NSBundle*) theBundle
{
    i f (g_p l u g i n B u n d l e) {
        r e t u r n N O;
    }
    g_p l u g i n B u n d l e = [theBundle r e t a i n];
    r e t u r n Y E S;
}

// Called to terminate our class, currently just to release our bundle
+ (void) t e r m i n a t e P l u g i n C l a s s
{
    i f (g_p l u g i n B u n d l e) {
        [g_p l u g i n B u n d l e r e l e a s e];
        g_p l u g i n B u n d l e = n i l;
    }
}
```

```
}  
}
```

Also in `MyNSBP_App's -awakeFromNib` is a call to the plug-in's `+instantiatePlugin` class method. This is a class method that returns the actual plug-in object. The method simply allocates and initializes an `MyNSBP_Plugin` object and loads its associated nib file, `RemoveColor.nib`. There is nothing in the nib file that we are using, but if there were, we could implement `-awakeFromNib` in the plug-in as well to access those items (see [Listing 4.10](#)).

Listing 4.10 `MyNSBP_RemoveColor +instantiatePlugin`

```
// Called to instantiate our plugin, currently to alloc, init and load the nib  
+ (id)instantiatePlugin  
{  
    MyNSBP_RemoveColor* instance =  
        [[[MyNSBP_RemoveColor alloc] init] autorelease];  
    if (instance && [NSBundle loadNibNamed:@"RemoveColor" owner:instance]) {  
        return instance;  
    }  
    return nil;  
}
```

The plug-in's `-doPlugin:` method (see [Listing 4.11](#)) is called when the user selects the plug-in from the Filter menu. As you remember, it is passed an `NSImage` and expected to alter it and return the altered version to the host application. The first thing this plug-in does is ask the user what color to remove by showing the window in [Figure 4.18](#). We will look at how the `-askUserWhatColorToRemove` method works in a moment, but for now know that it returns an integer of `0` if None was selected, `1` for Red, `2` for Green, or `3` for Blue.

Listing 4.11 `MyNSBP_RemoveColor -doPlugin:`

```
// Called to alter the image  
- (NSImage*)doPlugin:(NSImage*)theImage  
{  
    int whichColor = [self askUserWhatColorToRemove];  
  
    // If the user chose a color (other than None)  
    if (whichColor) {  
        NSRect theRect;  
  
        // Calculate theRect of the image  
        theRect.origin.x = theRect.origin.y = 0;  
        theRect.size = [theImage size];  
  
        // Lock the focus, draw, and unlock  
        [theImage lockFocus];  
  
        // Set the proper color  
        if (whichColor == 1) { // red  
            [[NSColor redColor] set];  
        } else if (whichColor == 2) { // green  
            [[NSColor greenColor] set];  
        }  
    }  
}
```

```

    } else if (whichColor == 3) { // blue
        [[NSColor blueColor] set];
    }

    // Fill the rect with the NSCompositePlusDarker mode to remove the color
    NSRectFillUsingOperation(theRect, NSCompositePlusDarker);

    // Unlock the focus
    [theImage unlockFocus];
}

// Return the image to the caller (altered or not)
return theImage;
}

```

Assuming that a color was chosen, this method prepares for the alteration by calculating the rectangle of the image, locking the focus, and setting the proper color for drawing. Then, using the `NSRectFillUsingOperation` function and the `NSCompositePlusDarker` drawing mode, the rectangle is filled. This mode removes the color that is currently set from the image. This is much quicker than looking at each pixel and removing the green component by hand. Once complete, we unlock the focus and return the image—albeit with no green.

Note

`NSRectFillUsingOperation` is one of many drawing routines in `NSGraphics.h`. Take a look at the contents of this header file for a lot of useful functions.

The `-askUserWhatColorToRemove` method (see [Listing 4.12](#)) is a handy one to have around because it shows what it takes to display a modal window from within an application (or in this case, from within a plug-in in an application). The first thing this method does is allocate and initialize our `SettingsController` object. This causes the `-initWithWindowNibName:` method to be called from the `SettingsController`'s `-init` method (see [Listing 4.13](#)). At this point, the `SettingsController` and the nib file associated with it are both loaded and ready to be used.

Listing 4.12 `MyNSBP_RemoveColor -askUserWhatColorToRemove`

```

// Ask the user what color to remove
- (int) askUserWhatColorToRemove
{
    int whichColor = 0;

    // Allocate our settings dialog box
    SettingsController *sc = [[SettingsController alloc] init];

    if (sc) {

        // Run the dialog box modal. The SettingsController will call
        // stopModalWithCode with the tag of the button that was pressed
        // to end the modal loop.
        [sc showWindow: self];
    }
}

```

```

    whi chCol or = [NSApp runModal ForWi ndow: [sc wi ndow]];

    // Deallocate the preference controller, we no longer need it
    [sc release];
    sc = nil;

}

// return the chosen color, or 0 (none)
return whi chCol or;
}

```

Assuming that the `SettingsController` was allocated without error, we call its `-showWindow:` method, which displays the modal window. This works because in Interface Builder the window is connected to the window outlet of the File's Owner, which is the `SettingsController`. We then call `NSApplication`'s `-runModalForWindow:` method passing the window in question. This method will manage the window modally until the user clicks one of the buttons in the window—all of which are connected to the `-myDoneAction:` method in `SettingsController` (refer to [Listing 4.11](#)).

Listing 4.13 SettingsController Implementation

```

@implementation SettingsController

- (id)init
{
    // Load the Settings.nib file
    self = [super initWithWindowNibName:@"Settings"];
    return self;
}

- (IBAction)myDoneAction: (id) sender
{
    [NSApp stopModalWithCode: [sender tag]];
}

@end

```

When a button is pressed and `-myDoneAction:` is triggered, `NSApp`'s `-stopModalWithCode:` method is called passing the tag of the button that was clicked—either `0`, `1`, `2`, or `3`. Back in `-askUserWhatColorToRemove`, the `SettingsController` is released and the selected value is returned. This is one way you can implement simple modal windows in Cocoa!

A Word About Symbol Collisions

Whenever you have more than one code base intended to work together in a way such as outlined in this chapter, you need to be concerned with symbol collisions. Symbol collisions are when you have defined the same symbol more than once and they cause a problem. A symbol can be a C function or variable name, classnames, formal protocol names, and class/category name pairs. For example, if two separate plug-ins have a class named `MyController`, when the second one loads, you might find that it causes problems in the host application when the symbols collide.

You see, the Objective-C runtime dislikes symbol collisions. In the past, it would not load code from a bundle if it defined a symbol that had already been loaded by an application, library, or another plug-in. Since Mac OS X 10.2, this has been relaxed somewhat; the runtime will try its best to keep your program limping along without any guarantee that it will succeed.

However, in the case of the `MyController` class, a host of problems might occur. The second class loaded might not implement the same methods, causing a "doesn't respond to method" error when the class is called. It might implement a method of the same name, but return different results. In one of the worst cases, it might take less (or more) memory to store the instance variables of the class and might cause your program to crash when they are read or written to.

The best way to protect C-style symbols (functions, variables, and so on) from interfering with other code elements is to define them as `static` or `private extern`. That is, limit the scope of the symbol either to the file in which it is defined or the file in which it is linked.

The only way to protect Objective-C symbols (classes, protocols, class/category name pairs, and so on) is to use relatively unique symbol name prefixes. One method is to prefix your symbols with the same unique identifier. For example, if you were writing a screen effect that displayed pictures of the beach, you might name your classes `BeachView`, `BeachSettings`, and so on. A better way would be to use the reverse domain prefix—such is done with property list files throughout this book, but that can be cumbersome. Can you imagine having classes named `ComTripl eSoftBeachView` and `ComTripl eSoftBeachSettings`?

So, the good news is that Mac OS X 10.2 is better about these things than previous releases, but it's still not foolproof. NSBundle-based plug-in authors need to be more aware of the risks of symbol collisions in their bundle code and know how to protect themselves from it.

Try This

Here are some ideas to expand the plug-in projects in this chapter; try these on your own.

Find a search engine that is not supported by the plistPlugin application and create a new plug-in for it. Add the plug-in to the application without recompiling it.

Write a new filter plug-in for the image editing application. Try writing a plug-in that removes all the pixels that are a specific color in the image. Use what you've learned in [Chapter 3](#), "Cocoa Applications," to allow the user to choose a color and to access the individual pixels in the image.

Make the Remove Color window in [Figure 4.14](#) a sheet so that it does not lock up the application while it is selected. You will need to pass the main application window in to the plug-in in order to accomplish this, via `-doPlug:`. You might consider renaming this method `-filterImage:window:`.

Did you know that you could actually build multiple project pieces in the same project file? Try retrofitting the `NSBundle` project in this chapter to build both the application and all the plug-ins in one single Project Builder project file. You will essentially create multiple Targets in Project Builder. This is a good way to implement code if you are writing all the pieces, but our example provides for third parties to write the plug-ins while someone else, such as yourself, writes the application.

Conclusion

Congratulations! This was a very complex chapter to get through in one piece. Even if the concept of plug-ins isn't new to you, implementing them in the ways described here might have been. Using Cocoa and Objective-C is a new paradigm for many Macintosh developers of old. It's okay to be a little confused as you weave through the maze that is OS X software development. The point is, in this chapter, you have been given working examples of two plug-in architectures that you can use in your own development projects. In the next chapter, I will show yet another plug-in architecture for those of you supporting Carbon-based applications.

Chapter 5. Carbon Plug-ins

"I got creative and ambitious—a dangerous combination."

—Unknown (from CocoaDev Mailing List)

What did developers do before Cocoa? They ate, drank, and slept Carbon—that's what they did! Carbon is a set of APIs that bridge the gap between previous versions of the Mac OS and Mac OS X. If you've programmed the Macintosh in the past, many of the APIs will look familiar to you: File Manager, Memory Manager, and Resource Manager. Both Carbon and Cocoa applications can access the Carbon APIs, which is an advantage for developers pursuing Cocoa development.

Depending on the last time you wrote a Macintosh application, things might have changed. A nib-based Carbon application used in Project Builder can literally be as simple as the code in [Listing 5.1](#). It probably looks different from what you remember in THINK C.

Listing 5.1 Carbon Application Main

```
int main(int argc, char* argv[])
{
    INibRef      nibRef;
    WindowRef    window;
    OSStatus     err;

    // Create a Nib reference passing the name of the nib file
    // (without the .nib extension)
    // CreateNibReference only searches in the application bundle.
    err = CreateNibReference(CFSTR("main"), &nibRef);
    require_noerr(err, CantGetNibRef);

    // Once the nib reference is created, set the menu bar.
    // "MainMenu" is the name of the menu bar object.
    // This name is set in InterfaceBuilder when the nib is created.
    err = SetMenuBarFromNib(nibRef, CFSTR("MenuBar"));
    require_noerr(err, CantSetMenuBar);

    // Then create a window.
    // "MainWindow" is the name of the window object.
    // This name is set in InterfaceBuilder when the nib is created.
    err = CreateWindowFromNib(nibRef, CFSTR("MainWindow"), &window);
    require_noerr(err, CantCreateWindow);

    // We don't need the nib reference anymore.
    DisposeNibReference(nibRef);

    // The window was created hidden so show it.
    ShowWindow(window);

    // Call the plugin
    CallPlugin();
}
```

```
// Call the event loop
RunAppl i cati onEventLoop();
```

```
CantCreat eWi ndow:
CantSetMenuBar:
CantGetNi bRef:
    return err;
}
```

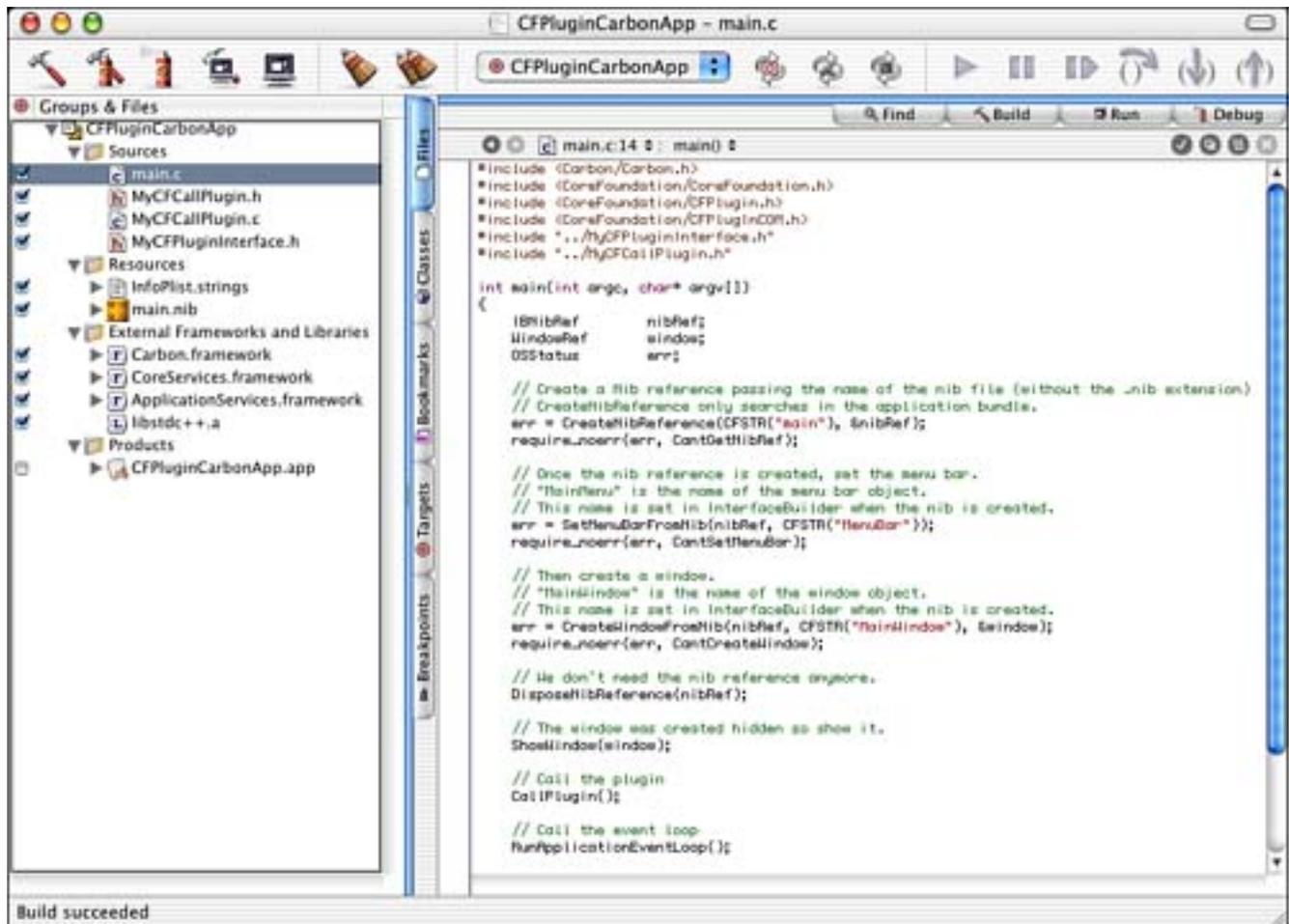
Mind you, this application doesn't actually do anything except display a simple menu bar and an empty window. These items are pulled from a nib file as created in Interface Builder. [Figure 5.1](#) shows the Starting Point choices that Interface Builder offers when creating a new nib file. Our application uses the Main Window with Menu Bar. You'll note that once our application initializes everything, it makes a single function call to `RunAppl i cati onEventLoop` that handles everything else.

Figure 5.1. Choosing a nib file type in Interface Builder.



You've learned that Carbon applications can make use of nib files. This allows developers of these applications to avoid having to use ResEdit (you remember ResEdit, don't you) to manage the resources of their application. You can simply create your windows and dialogs directly in Interface Builder. The entire project in Project Builder for our simple Carbon application is as shown in [Figure 5.2](#). Project Builder contains a template, Carbon Application (Nib Based), that we used to start this project.

Figure 5.2. A Carbon application project in Project Builder.



The Results

You've seen one project already, but this chapter actually has three projects that come out of it. The primary project is the **CFPlugin**-based Carbon plug-in itself. The other two are a Carbon application (which you've seen previously) and a Cocoa application that call the plug-in. Both applications call the exact same C language function that calls the exact same plug-in. The applications themselves are nothing more than quick and dirty wrappers to demonstrate the process. We'll step through them later in this chapter.

[Figures 5.3](#) and [5.4](#) show what each application looks like when calling the plug-in. The window to the left is part of the application. The alert window to the right is called from within the plug-in itself. The "**flag = YES**" display is showing the value of a Boolean value passed into the plug-in from the calling application. Note that both applications are very similar: the only difference is that the Carbon application calls the plug-in immediately upon launching, and the Cocoa application has a button that must be clicked to call the plug-in. There is no reason for this other than it's easier to create a button in Cocoa than it is in Carbon, so we decided to mix it up a bit and make sure that you were paying attention.

Figure 5.3. A Carbon application calling a Carbon plug-in.

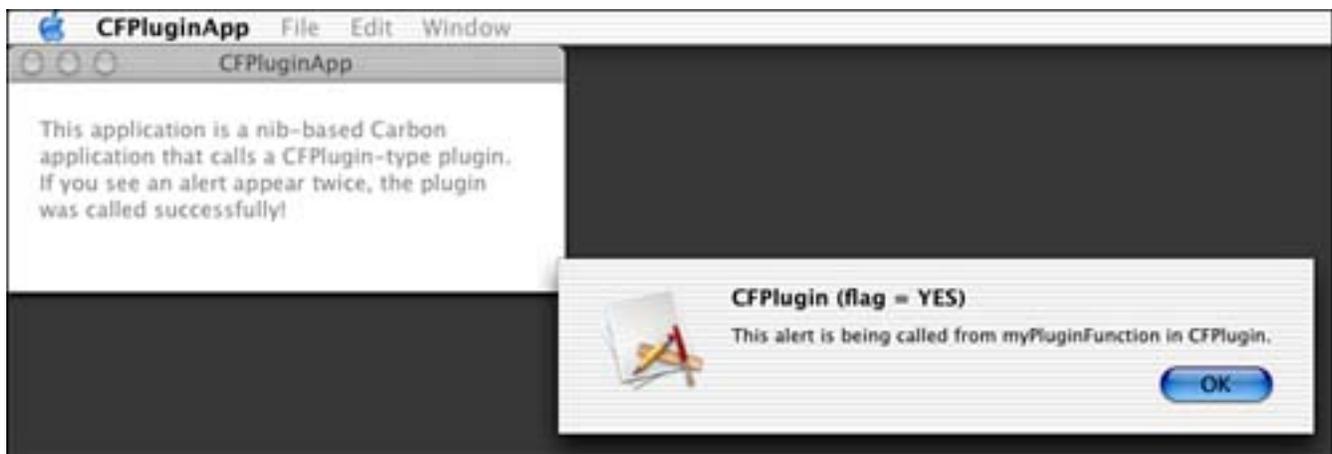


Figure 5.4. A Cocoa application calling a Carbon plug-in.



Before we look at the plug-in project, here's a few words about how CFPlugins work at a very high level.

The basic tenants of the plug-in model are as follows:

- You define one or more plug-in types.
- Each plug-in type consists of one or more plug-in interfaces.
- Each plug-in implements all functions in all interfaces for all types that the plug-in supports.
- Each plug-in also provides factory functions that create each type it implements.
- When a host application loads a plug-in, it can call any of the factory functions within the plug-in to instantiate any of the types that the plug-in supports. The application also receives a pointer to the **IUnknown** interface to the type it instantiated.
- The host application then uses the **IUnknown** interface to query the implemented types for their function tables. Once this is obtained, the host application can call the unique functions for the types that are implemented within the plug-in.

An example might be an audio-editing plug-in. The type of plug-in is Audio Editor. We might then have two interfaces—one for Real-Time audio processing and one for File-based audio processing. Because our plug-in supports both interfaces, we will implement all functions that pertain to Real-Time audio processing and File-based audio processing. We will provide one factory function to create the Audio Editor type. The host application would then use the factory function to obtain the **IUnknown** interface for the type and then use the `QueryInterface` function to obtain the actual interface with function pointers to the unique functions in each implementation. The last thing to do is call the function in the plug-in—`processRealTimeAudioEase`.

Keep these points in mind as we walk through the code of the plug-in and the host application, and you will see each of them accounted for. They might seem confusing now, but when you see all the pieces fit together, it will make much more sense.

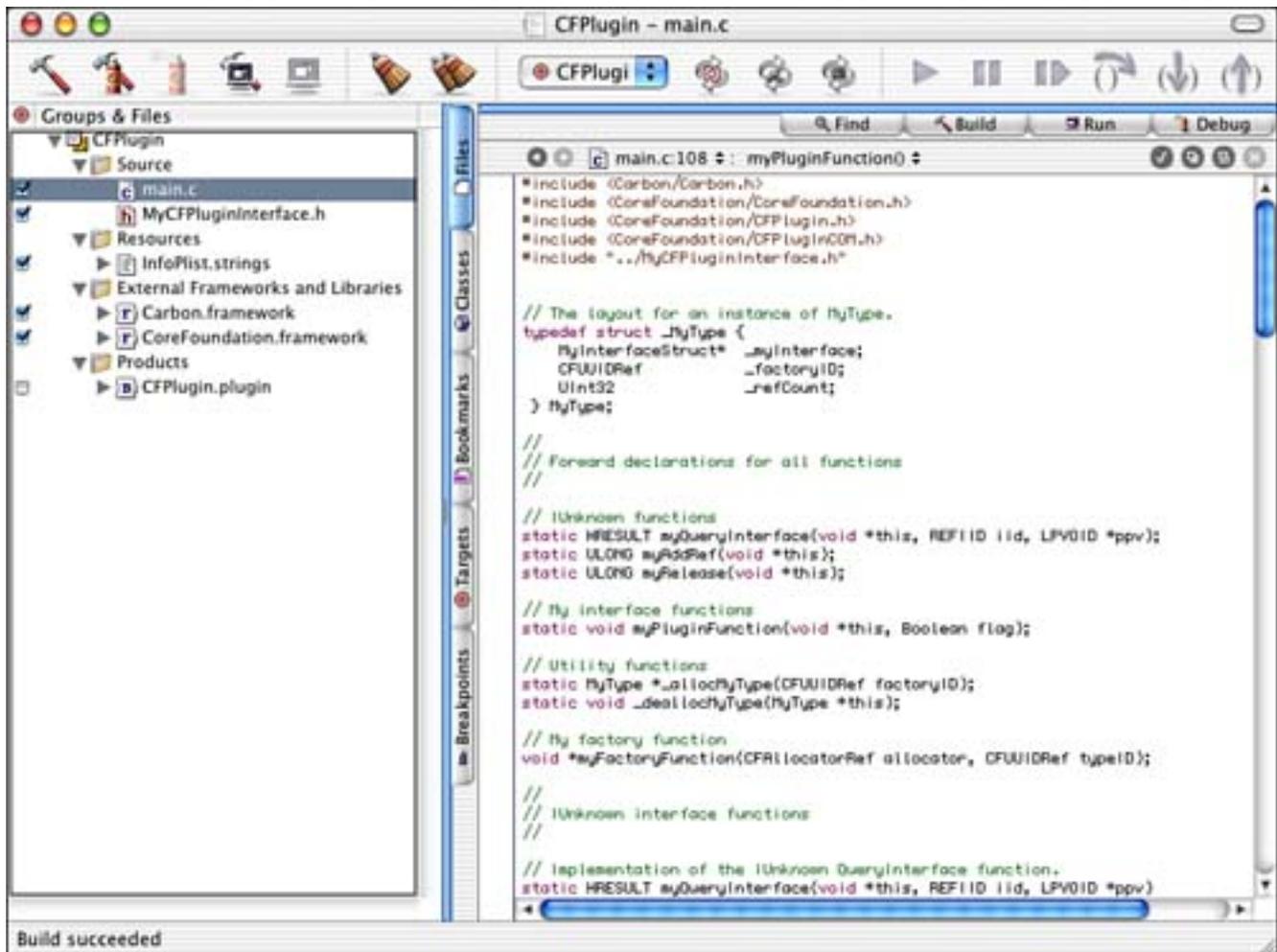
[\[Team LiB \]](#)

◀ PREVIOUS NEXT ▶

The CFPlugin Project

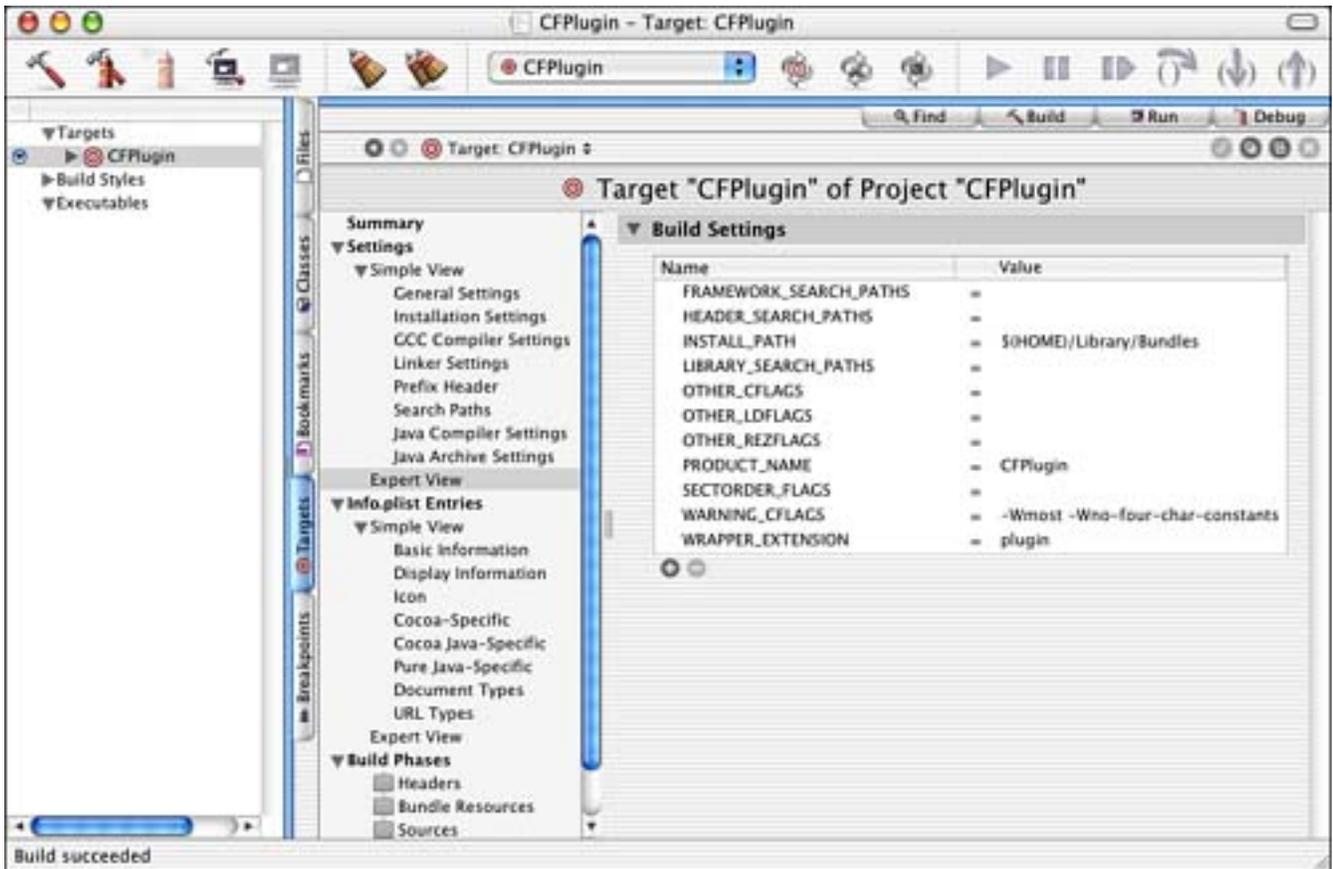
The primary point of this chapter is to discuss the **CFPlugin**-based Carbon plug-in, so let's dive in. In Project Builder, we started with the **CFPlugin** Bundle project template. This didn't really give us any default source code, but it set up a few things in the project settings for us. Note that the project file in [Figure 5.5](#) is not overcrowded. Let's verify a few settings before we continue.

Figure 5.5. The CFPlugin project in Project Builder.



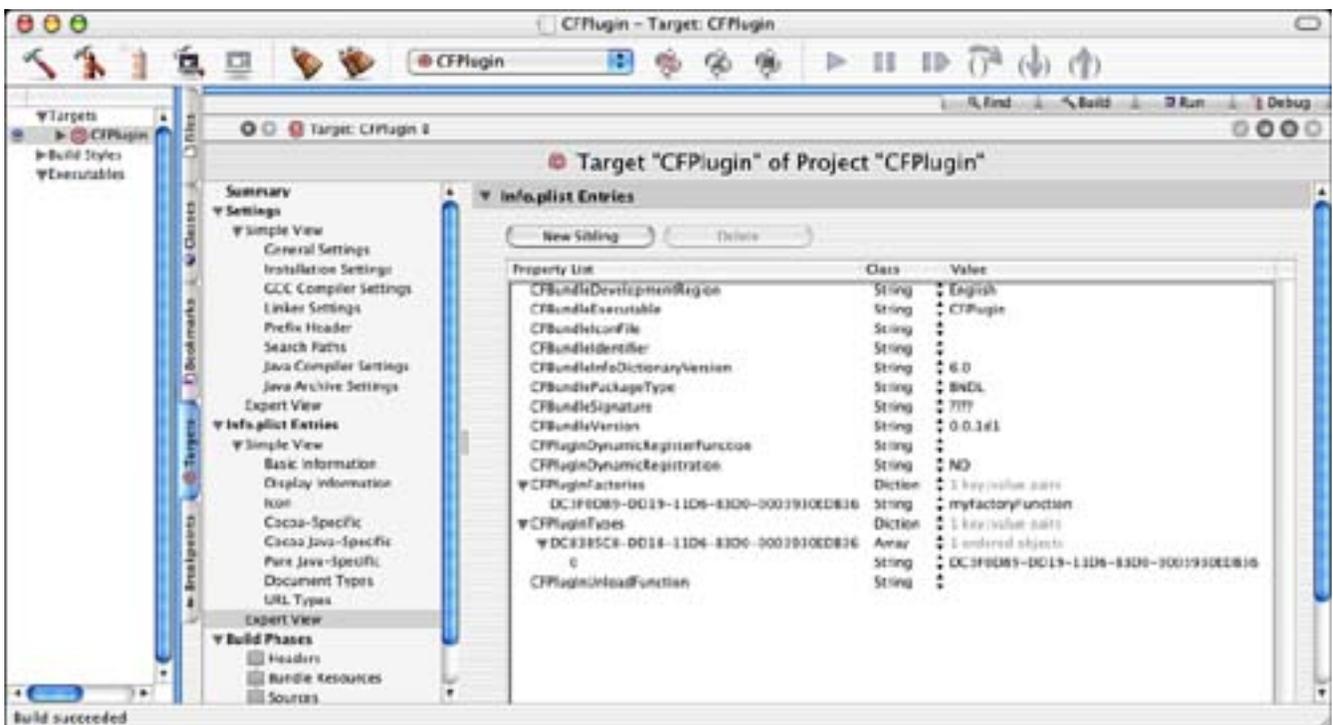
The first item you will want to verify is the **WRAPPER_EXTENSION** build setting (see [Figure 5.6](#)). Although it makes little difference what this extension is set to, "plugin" is the standard for this type of project. You will want to make sure that this is set accordingly.

Figure 5.6. The CFPlugin project Target build settings in Project Builder.



The `Info.plist` entries (see [Figure 5.7](#)) are a bit more complex for this project than any other we have worked with up to this point.

Figure 5.7. The CFPlugin project Info.plist entries in Project Builder.



You will notice the standard entries, but a few are new to you here. Let's look at the new items in the order

that they appear:

- `CFPlugInDynamicRegistrationFunction` contains the name of a custom function to be called to perform dynamic registration. This is only called if `CFPlugInDynamicRegistration` is set to `YES`. If `CFPlugInDynamicRegistration` is set to `YES` and this key is not defined, a function named `CFPlugInDynamicRegistration` is called. This function is defined as follows: `typedef void (*CFPlugInDynamicRegistrationFunction) (CFPlugInRef plugIn);`
- `CFPlugInDynamicRegistration` determines how the plug-in wants to be registered. If the value is `NO`, static registration is used. If the value is `YES`, dynamic registration is used. We will be using static registration in this project.
- `CFPlugInFactories` is used for static registration. Its value is a dictionary whose keys are factory UUIDs (Universally Unique Identifiers) in string format and whose values are function names within the plug-in. Factories are functions within the plug-in that are used to create instances of the plug-in. A plug-in can have multiple factories that create multiple types of plug-ins; that is, a plug-in can actually contain multiple plug-ins. This function is defined as follows: `typedef void (*CFPlugInFactoryFunction) (CFAllocatorRef allocator, CFUUIDRef typeUUID);`
- `CFPlugInTypes` is used for static registration. Its value is a dictionary whose keys are type UUIDs in string format and whose values are arrays of factory UUIDs. You are essentially linking the factories to the types they create in this entry.
- `CFPlugInUnloadFunction`, although not listed here, can be used to define a custom function that is called when the plug-in's code is unloaded. If it is not defined, no function is called. This function is defined as follows: `typedef void (*CFPlugInUnloadFunction) (CFPlugInRef plugIn);`

Note

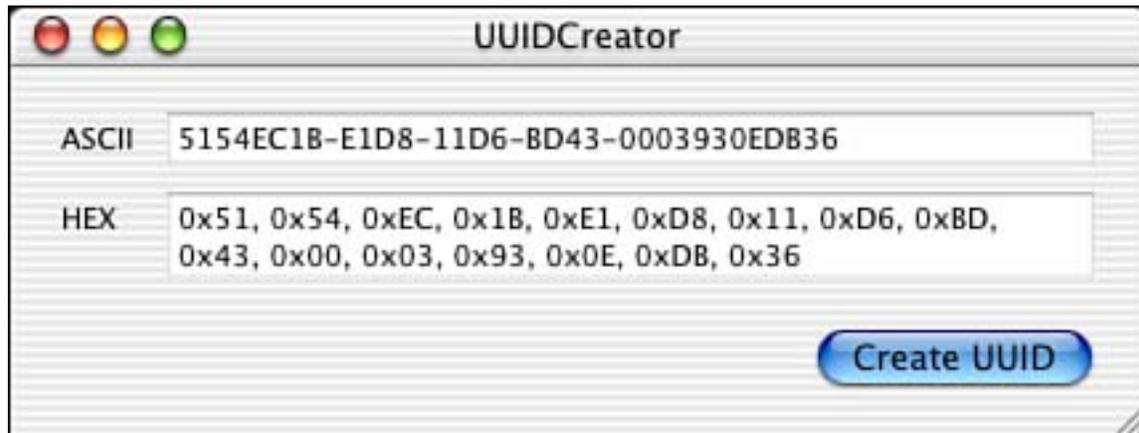
As you can see by my example, I will be using static registration to demonstrate the `CFPlugIn` architecture. However, you can implement dynamic registration if your plug-in has the need to check for specific hardware configuration, and so on. Dynamic registration allows you to create your factory/type relationships in code at runtime as opposed to in a static `Info.plist` entry. The use of dynamic registration loads the code of the plug-in immediately because the `CFPlugInDynamicRegistrationFunction` must be called. With static registration, however, the code doesn't need to be loaded until the host application needs to call it—which is a good thing with regards to memory management.

What Is a UUID?

You will notice that we are making use of something called a UUID to describe our interface, factories, and types. A UUID is a Universally Unique Identifier, which is essentially a value that is unique across all computers in the entire world. It is a 128-bit value that combines a value unique to the computer, usually a hardware Ethernet address, and a value representing the number of 100-nanosecond intervals since October 15, 1582 at 00:00:00.

You can use the UUIDCreator application (see [Figure 5.8](#)), written especially for this book, to generate these numbers for you. You will notice that UUIDCreator creates UUIDs in both ASCII and HEX formats. You need to use the ASCII format in your property lists, whereas in code, you will tend to use the HEX version more. The software and source code is available on this book's Web site, along with all the source code for every project.

Figure 5.8. The UUIDCreator application.



The CFPlugin Project Source Code

Now that the project settings are out of the way, let's dive into the source code. The good news is that there is only one source file with less than a dozen functions in it. The bad news is that it can be very confusing, so put on your thinking cap.

The **CFPlugin** architecture is based on Microsoft's COM (Component Object Model) technology. This means that, in theory, you can write plug-ins that are cross-platform and compatible with any host that supports COM. We are not going to worry about that right now though; we're going to concentrate on getting one working under OS X first.

The Plug-in Interface

The first thing a plug-in needs is an interface (see [Listing 5.2](#)). The plug-in interface is used by the application to access the plug-in's functionality. It also defines the UUIDs for the types, factories, and the interface itself. Notice that the definitions are made using the HEX format of the UUID, beginning with a **NULL**.

Listing 5.2 Plug-in Interface in `MyCFPluginInterface.h`

```
// Define the UUID for the type
#define kMyTypeID (CFUUIDGetConstantUUIDWithBytes(NULL, 0xDC, 0x83, 0xB5,
0xC8, 0xDD, 0x18, 0x11, 0xD6, 0xB3, 0xD0, 0x00, 0x03, 0x93, 0x0E, 0xDB, 0x36))

// The UUID for the factory function
#define kMyFactoryID (CFUUIDGetConstantUUIDWithBytes(NULL, 0xDC, 0x3F, 0x0D,
0x89, 0xDD, 0x19, 0x11, 0xD6, 0xB3, 0xD0, 0x00, 0x03, 0x93, 0x0E, 0xDB, 0x36))

// Define the UUID for the interface
// MyType objects must implement MyInterface
#define kMyInterfaceID (CFUUIDGetConstantUUIDWithBytes(NULL, 0x07, 0x1D, 0x88,
0x1D, 0xDD, 0x1A, 0x11, 0xD6, 0xB3, 0xD0, 0x00, 0x03, 0x93, 0x0E, 0xDB, 0x36))

// The function table for the interface
typedef struct MyInterfaceStruct {
    IUNKNOWN_C_GUTS;
    void (*myPluginFunction)(void *this, Boolean flag);
} MyInterfaceStruct;
```

Also in the interface is a structure that defines the functions in our plug-in. In this case, we only have one function, `myPluginFunction`, which returns `void` and accepts two parameters. The first parameter, `this`, is a pointer to the interface itself. The second parameter, `flag`, is a Boolean that is displayed in the dialog box that the plug-in displays. You can define as many functions as you need using whatever parameters you require.

Also in the interface structure is something called `IUNKNOWN_C_GUTS`. `IUNKNOWN_C_GUTS` is a macro defined in `CFPluginCOM.h`. It defines the bare minimum needed in the function table structure `MyInterfaceStruct`. `IUnknown` can be considered the "base" interface that all COM objects are modeled on (the super class if you will). This means that all COM objects are really `IUnknown` objects at heart and all start with the three

functions `QueryInterface`, `AddRef`, and `Release` as shown in [Listing 5.3](#).

Listing 5.3 IUNKNOWN_C_GUTS in `CFPlugInCOM.h`

```
#define IUNKNOWN_C_GUTS \
    void *_reserved; \
    HRESULT (STDMETHODCALLTYPE *QueryInterface) \
    (void *thisPointer, REFIID iid, LPVOID *ppv); \
    ULONG (STDMETHODCALLTYPE *AddRef)(void *thisPointer); \
    ULONG (STDMETHODCALLTYPE *Release)(void *thisPointer)
```

Note

To gain more insight into the magic of COM, be sure to browse the header file `CFPlugInCOM.h`. This is where `IUnknown` is defined, as well as many other useful tidbits.

The Plug-in Main Function

Let's look at the source code that is our Carbon plug-in. Everything in this section can be found in `main.c` in the `CFPlugIn` project, as shown in [Figure 5.5](#).

You've seen the function table for our plug-in interface in [Listing 5.2](#). The interface needs a type to implement it. Our type could be considered an object because it contains both data and functions (via the interface). When we ultimately allocate and initialize our type of object, we want to return a complete unit as shown in [Listing 5.4](#).

Listing 5.4 MyType Definition in `main.c`

```
// The layout for an instance of MyType
typedef struct _MyType {
    MyInterfaceStruct*    myInterface;
    CFUUIDRef             factoryID;
    UInt32                refCount;
} MyType;
```

The first function that will be called in our plug-in is our factory function (see [Listing 5.5](#)). Shortly after creating the plug-in, the host application will find all factories in the plug-in that pertain to a particular object type that the application is interested in. The factory functions returned are directly related to the definitions in the `Info.plist` entries in [Figure 5.7](#).

Listing 5.5 myFactoryFunction in `main.c`

```
// Implementation of the factory function for this type
void *myFactoryFunction(CFAllocatorRef allocator, CFUUIDRef typeId)
{
    printf("myFactoryFunction\n");
}
```

```

// If correct type is being requested,
// allocate an instance of MyType and return the IUnknown interface
if (CFEqual(typeID, kMyTypeID)) {
    MyType *result = _allocMyType(kMyFactoryID);
    return result;
} else {
    // If the requested type is incorrect, return NULL
    return NULL;
}
}

```

When the host application calls `CFPlugInInstanceCreate` and causes our factory function to be called, we verify that the caller wants to create a type that we know about. Once validated, we create the type by calling the private `_allocMyType` function and return the result, which will be a pointer to a `MyType` structure as shown in [Listing 5.4](#). Note the `printf` function used for debugging purposes and to help trace our code at runtime.

Listing 5.6 `_allocMyType` in `main.c`

```

// Utility function that allocates a new instance
static MyType *_allocMyType(CFUUIDRef factoryID)
{
    // Allocate memory for the new instance
    MyType *newOne = (MyType*) malloc(sizeof(MyType));

    // Point to the function table
    newOne->_myInterface = &myInterfaceFtbl;

    // Retain and keep an open instance refcount for each factory
    newOne->_factoryID = CFRetain(factoryID);
    CFPlugInAddInstanceForFactory(factoryID);

    // This function returns the IUnknown interface so set the refCount to one
    newOne->_refCount = 1;
    return newOne;
}

```

[Listing 5.6](#) shows the `_allocMyType` function that actually allocates and initializes our type structure. Note that it fills the `myInterface` variable with a pointer to `myInterfaceFtbl`, shown in [Listing 5.7](#). `myInterfaceFtbl` is a static declaration of `MyInterfaceStruct` that contains a `NULL` pad, the "my" versions of the standard `IUnknown` functions (`QueryInterface`, `AddRef`, and `Release`), and the pointer to our one and only unique plug-in function, `myPlugInFunction`. Note that this static declaration mirrors the `MyInterfaceStruct` in [Listing 5.2](#) and the `IUNKNOWN_C_GUTS` in [Listing 5.3](#). We also keep track of the factory UUID so that we can follow the usage status of our type.

Listing 5.7 `myInterfaceFtbl` in `main.c`

```

// The MyInterface function table
static MyInterfaceStruct myInterfaceFtbl = {
    NULL, // Required padding for COM
    myQueryInterface, // These three are the required COM functions
}

```

```

myAddRef,
myRelease,
myPluginFunction }; // Interface implementation (specific to my plugin)

```

At this point, the host will have a copy of the `IUnknown` interface and can use the `QueryInterface` function to obtain the interface that is my "subclass" of `IUnknown`—the "my" interface. When `QueryInterface` is called, my plug-in's `myQueryInterface` function (see [Listing 5.8](#)) is subsequently executed.

Listing 5.8 `myQueryInterface` in `main.c`

```

// Implementation of the IUnknown QueryInterface function
static HRESULT myQueryInterface(void *this, REFIID iid, LPVOID *ppv)
{
    HRESULT hResult = E_NOINTERFACE;

    printf("myQueryInterface\n");

    // Create a CoreFoundation UUIDRef for the requested interface
    CFUUIDRef interfaceID = CFUUIDCreateFromUUIDBytes(NULL, iid);

    // Test the requested ID against the valid interfaces
    if (CFEqual(interfaceID, kMyInterfaceID)) {

        // If the MyInterface was requested, bump the ref count,
        // set the ppv parameter equal to the instance, and return good status
        // This calls through to myAddRef
        ((MyType*)this)->_myInterface->AddRef(this);
        *ppv = this;
        hResult = S_OK;

    } else if (CFEqual(interfaceID, IUnknownUUID)) {

        // If the IUnknown interface was requested, bump the ref count,
        // set the ppv parameter equal to the instance, and return good status
        // This calls through to myAddRef
        ((MyType*)this)->_myInterface->AddRef(this);
        *ppv = this;
        hResult = S_OK;

    } else {

        // Requested interface unknown, bail with error
        *ppv = NULL;
        hResult = E_NOINTERFACE;

    }

    // Release interface
    CFRelease(interfaceID);
    return hResult;
}

```

The `myQueryInterface` function converts the incoming interface UUID to a format that it can easily compare against using the `CFUUIDCreateFromUUIDBytes` function. It then validates if the requested

interface is one that is supported here. If so, we increment the reference count by calling `AddRef` (which really calls through to `myAddRef` in [Listing 5.9](#)) and return the `this` pointer, the actual plug-in interface.

Listing 5.9 `myAddRef` in `main.c`

```
// Implementation of reference counting for this type
// Whenever an interface is requested, bump the refCount for
// the instance NOTE: returning the refcount is a convention
// but is not required so don't rely on it
static ULONG myAddRef(void *this)
{
    printf("myAddRef\n");

    return ((MyType*) this)->_refCount++;
}
```

At this point, the host application will release the `IUnknown` interface by calling `Release`, which will subsequently call our `myRelease` function in [Listing 5.10](#).

Listing 5.10 `myRelease` in `main.c`

```
// When an interface is released, decrement the refCount
// If the refCount goes to zero, deallocate the instance
static ULONG myRelease(void *this)
{
    printf("myRelease\n");

    ((MyType*) this)->_refCount--;
    if (((MyType*) this)->_refCount == 0) {
        _deallocMyType((MyType*) this);
        return 0;
    } else
        return ((MyType*) this)->_refCount;
}
```

After decrementing the reference count, if it is zero, the type is deallocated by calling the private mirrored function of `_allocMyType`, `_deallocMyType` (see [Listing 5.11](#)).

Listing 5.11 `_deallocMyType` in `main.c`

```
// Utility function that deallocates the instance
// when the refCount goes to zero
static void _deallocMyType(MyType *this)
{
    CFUUIDRef factoryID = this->_factoryID;
    free(this);
    if (factoryID) {
        CFPlugInRemoveInstanceForFactory(factoryID);
        CFRelease(factoryID);
    }
}
```

The only thing left to do at this point is actually call the function that is unique to our type and interface. It's amazing how 95% of the source code is the setup of the mechanism to call this one simple function. The function in [Listing 5.12](#) is the entire reason for this chapter. When called, it will display a standard alert with the value of the flag variable displayed as `flag = YES` or `flag = NO`. The host application will ultimately release this interface as well when it is done with it.

Listing 5.12 `myPluginFunction` in `main.c`

```
// The implementation of the MyInterface function
static void myPluginFunction(void *this, Boolean flag)
{
    SInt16 outItemHit;
    StandardAlert(kAlertNoteAlert,
        flag ? "\pCFPlugin (flag = YES)" : "\pCFPlugin (flag = NO)",
        "\pThis alert is being called from myPluginFunction in CFPlugin.",
        NULL, &outItemHit);
}
```

Before you move on to see how the Carbon application makes this all come together, take a short break. Stretch, go look out a window, walk the dog, or play with the cat. When you return, we'll dive into the application source code. It's only one function, so it should be easy enough.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶


```

        (LPVOID *) (&interface));

    // Done with IUnknown
    // CFPlugin::myRelease is called here
    (*iunknown)->Release(iunknown);

    // If this is a "My" interface, try to call its function
    if (interface) {
        (*interface)->myPluginFunction(interface, TRUE);
        (*interface)->myPluginFunction(interface, FALSE);

        // Done with interface
        // This causes the plug-in's code to be unloaded
        // CFPlugin::myRelease is called here
        (*interface)->Release(interface);
    } else printf( "Failed to get interface.\n" );
    } else printf( "Failed to create instance.\n" );
} else printf( "Could not find any factories.\n" );

// Release the CFPlugin memory
CFRelease(plugin);

} else printf( "Could not create CFPluginRef.\n" );
}

```

Note

One thing you will note in this function is that we use `printf` to display error messages during development. If you want to leave these debugging statements in your code, you might consider using `fprintf` and printing your errors to standard error, such as this: `fprintf(stderr, "Could not blah!")`.

This function finds the plug-in explicitly. There is no enumeration through directories, although you could add that easily enough. We use a full path to the plug-in and pass it to the `CFURLCreateWithFileSystemPath` function. When you run this on your computer, you will need to change this path accordingly.

Once we have the path to the plug-in, we create a `CFPluginRef` from the URL. This is really nothing more than a bundle reference because our plug-in is really nothing more than a bundle. At this point, the plug-in's `Info.plist` entries are read in, and all the factories and types are registered with the operating system. If we were using dynamic registration, our dynamic registration function would be called at this point.

With the plug-in reference, we use the `CFPluginFindFactoriesForPluginType` function to see if the plugin implements the "my" type represented by the UUID `kMyTypeID`. If the array comes back with any items in it, we know that there is a factory to create this type. In our case, we only have one factory, but there might be any number. We then extract the factory UUID from the array and use it to get the `IUnknown` interface for the "my" type by calling the `CFPluginInstanceCreate` function. This is where our plug-ins factory function (refer to [Listing 5.5](#)) is called. If multiple factories were returned, we would simply loop through the array to access each of them in turn. We would then examine each one further to decide which

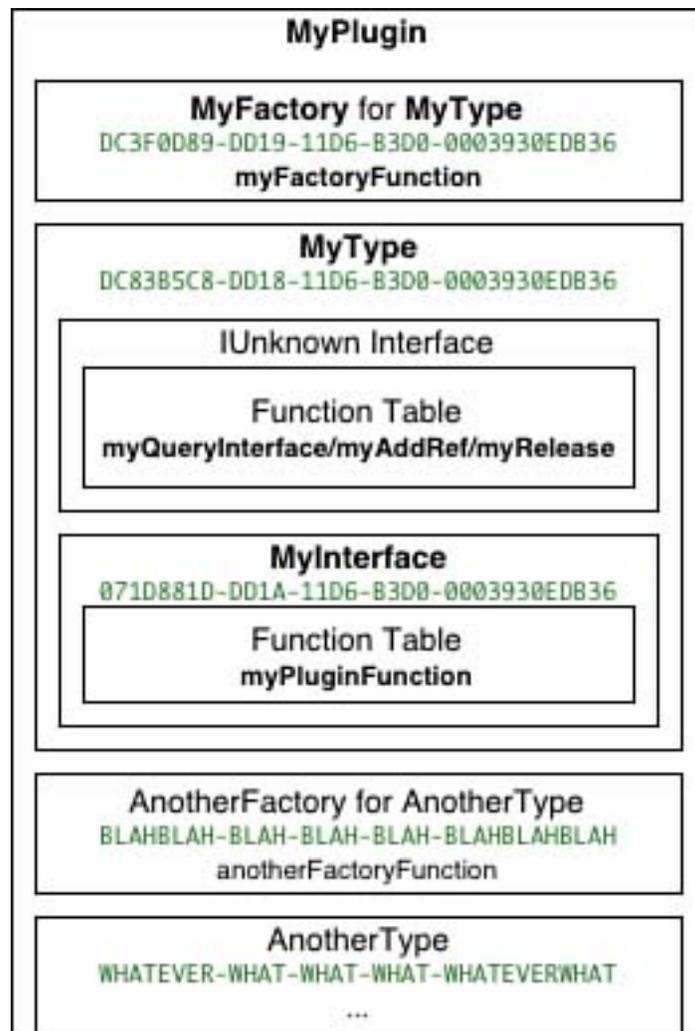
one to use, as follows.

After we obtain the **IUnknown** interface, we can further query the plug-in for the specific "my" interface by calling the **QueryInterface** function. This is where our plug-in's **myQueryInterface** (refer to [Listing 5.8](#)) is called. When an interface is returned, we are through with the **IUnknown** interface and can release it.

At this point, we have a pointer to the **MyInterfaceStruct** (the interface that our plug-in implements) that we can use to call the **myPluginFunction** passing whatever parameters we have defined. In our case, we pass the interface itself (**this**) and a Boolean value (**flag**). This is when you will see the alert appear from within the plug-in (refer to [Figure 5.3](#)). After we are done with the interface, we release it as well and then ultimately release the plug-in reference.

Let's look at a diagram that might help to solidify your understanding of how these pieces fit together (see [Figure 5.9](#)). Note that this model allows for an extremely complex set of relationships between the plug-ins and the host application, including the use of multiple factories for multiple types. You need not implement every possible complexity to use the architecture successfully. Don't over engineer for the sake of over engineering.

Figure 5.9. The CFPlugin architecture.



Note the relationships to the plug-in, the factory, the type, and the interface in [Figure 5.9](#). Multiple

interfaces are possible as are multiple types and multiple factories at any stage of this diagram. This is a simple example of what we have presented in this chapter.

A lot of steps and intricacies need to come together to make all this work properly. You might need to reread both the plug-in and the application sides again for it to all fit together. Be sure to look at the source code too.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The CFPluginCocoaApp Host Application Project

It is possible to call a **CFPlugin**-based Carbon plug-in from within a Cocoa application. There really isn't much to talk about here because the same **CallPlugin** function in [Listing 5.13](#) is called from within the Cocoa application to call the same plug-in developed earlier. It's code reuse at its finest!

The CallPlugin button (as you saw in [Figure 5.4](#)) is connected to the **doCallPlugin**: method in [Listing 5.14](#). When the button is pressed, the method is called, which does nothing more than call through to the **CallPlugin** function reviewed previously. This is not only an example of a Cocoa application calling a standard C function, but also a Cocoa application calling a **CFPlugin**-based Carbon plug-in.

Listing 5.14 **doCallPlugin** in **AppController.c**

```
- (IBAction)doCallPlugin:(id)sender
{
    CallPlugin();
}
```

Note

So, should you use Cocoa or Carbon when it comes to plug-ins? When I asked many professional Mac OS X developers this question, the answer came up as expected:

If you are supporting legacy code or operating systems in any way, you might need to use **CFPlugins**. This gives you the advantage of compatibility with pre-Mac OS X operating systems as well as the ability to unload unused code at the expense of a more complex architecture.

If you are developing brand new applications for OS X only and not looking back, use **NSBundle**. **NSBundle** provides an elegant interface to your plug-ins, but it is only compatible with Mac OS X and does not (currently) allow code to be unloaded when no longer needed.

Try This

Here are some ideas to expand the plug-in project in this chapter; try these on your own.

Load the Carbon plug-in from your own application bundle as in [Chapter 4](#) using directory enumeration. Don't forget to set up the project file to copy the plug-in into the PlugIns directory.

Choose your poison, Carbon or Cocoa, and write a useful application that makes use of the **CFPlugIn** mechanism. If you're brave, you might try writing the image-editing application from [Chapter 4](#) using **CFPlugIn** instead of **NSBundle**.

Conclusion

You now have a good platform to base your plug-in development on. Not only have you learned about calling Cocoa plug-ins from Cocoa in [Chapter 4](#), but also Carbon plugins from Carbon and Cocoa in this chapter. Throughout the chapters of this book, you will see plug-in type code implemented in various ways, proving that there truly are as many ways to implement something as there are programmers! Use the examples presented here to get you started, and be sure to share your improvements with the developer community.

Chapter 6. Frameworks

"You don't know what I know!"

—said the Framework to the Application

Frameworks provide a convenient way to distribute a library along with headers, documentation, and other resources. Frameworks are most analogous to libraries in C. If you're a C programmer, you have undoubtedly created or used a library. You might not have known it, but you have.

You might be familiar with libraries that come with a `.lib` file (the library itself) and a separate `.h` file (the header, or interface, file). The library file contains all the compiled source code to perform certain tasks, whereas the header file defines the interface to that code. By adding the library to your project and including the header file in your source files, you can access the functionality of the library without ever seeing its source code. Frameworks are very similar, but are much more advanced in structure and form.

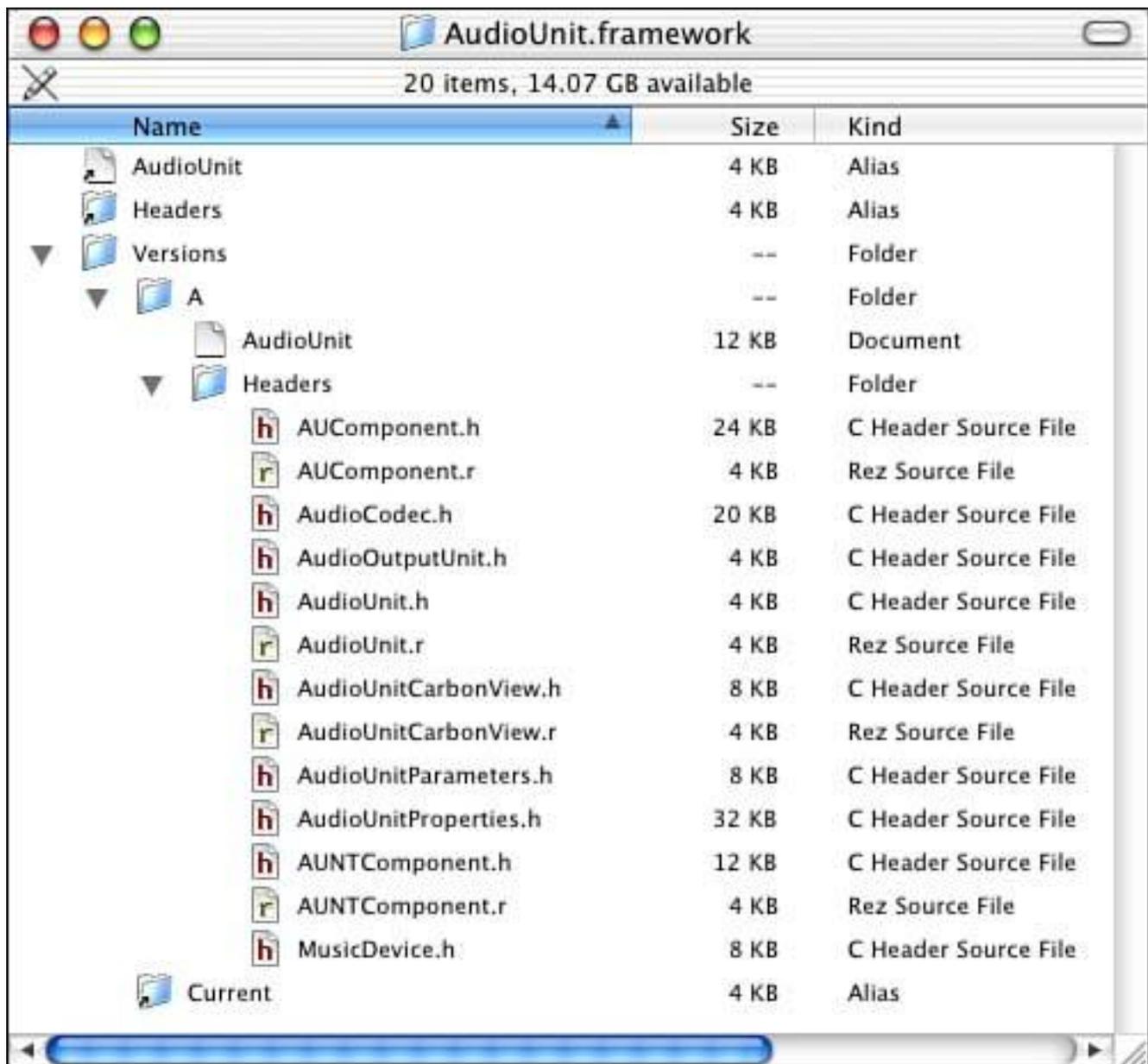
Note

I know it's early in the chapter, but this is just a quick note to say that although libraries ending in `.lib` are popular, there are plenty of others. Dynamic libraries end in `.dylib` and static libraries end in `.a`.

Similar to libraries, frameworks allow you to easily hide the implementation of classes or functions while providing an interface for others to access the hidden functionality. For example, you might have a highly secure encryption algorithm that someone else in your company needs to use, but you don't want the source code getting out. You can easily compile this function in a framework, release it with an interface file, and not have to worry. You can provide updates as needed and not be concerned about someone having old copies of your source code lying around. Frameworks can automatically handle versioning so that the latest version executes even if multiple versions are installed.

In addition, frameworks are bundles. Therefore, they can contain compiled code, plists, resources, nib files, documentation, and so on. You can place just about anything you like (including other frameworks) in a framework's bundle. Framework bundle names end in `.framework`. [Figure 6.1](#) shows what a framework looks like in the Finder.

Figure 6.1. Apple's AudioUnit framework.



Note that Apple's AudioUnit framework looks just like a folder in the Finder. You can open it up and browse the contents the same as you would any folder you create. The framework stores specific versions of its content in a Versions directory; you can use the Current alias (actually a symbolic link) to find the framework's current version. The Headers directory contains the public interface to the framework. To use this framework, you would simply add it to your project and include the required header files in your source code. The framework will work in either a Cocoa or Carbon project, as you will see shortly. [Listing 6.1](#) shows that the AudioUnit.h header includes all the other headers defined in the framework. Therefore, in many cases, you only need to be concerned about including the single file. Your project now has access to all the published functions in the framework.

Listing 6.1 AudioUnit.h

```
#ifndef __AUDIOUNIT__
#define __AUDIOUNIT__

// This is the main AudioUnit specification
#include <AudioUnit/AUComponent.h>
```

```
#i ncl ude <Audi oUni t/Audi oOut put Uni t. h>
#i ncl ude <Audi oUni t/Musi cDevi ce. h>

#i ncl ude <Audi oUni t/Audi oUni tPropert i es. h>
#i ncl ude <Audi oUni t/Audi oUni tParamet ers. h>

#i ncl ude <Audi oUni t/Audi oUni tCarbonVi ew. h>

#i ncl ude <Audi oUni t/Audi oCodec. h>

// Thi s fi l e r e l i e s o n A U C o m p o n e n t . h
// a n d c o n t a i n s t h e d i f f e r e n c e s o f V e r s i o n 1 A P I
#i ncl ude <Audi oUni t/AUNTC o m p o n e n t . h>

#endi f /* __AUDI OUNIT__ */
```

Note

If you've read any books in the past that discussed code fragments, you are already familiar with the capabilities of the Code Fragment Manager. One of the nice features of this manager was versioning. Code fragments would automatically run the most recent version of themselves if multiple versions were installed. Frameworks are designed in such a way that multiple versions of a framework can be made available and accessed accordingly. You can check to see what the current version of a Framework is by using the `CFBundleGetVersionNumber` function in `CFBundle.h`. This can be useful if you want to check for features or implement workarounds.

Normally your application will link against a specific version of a framework: "Link against version A of AppKit," for example. Using this mechanism, you can run older programs on newer computers since the older version of AppKit will still exist on the new machine. This topic is discussed in more detail in Apple's Mac OS X Overview documentation.

[\[Team LiB \]](#)

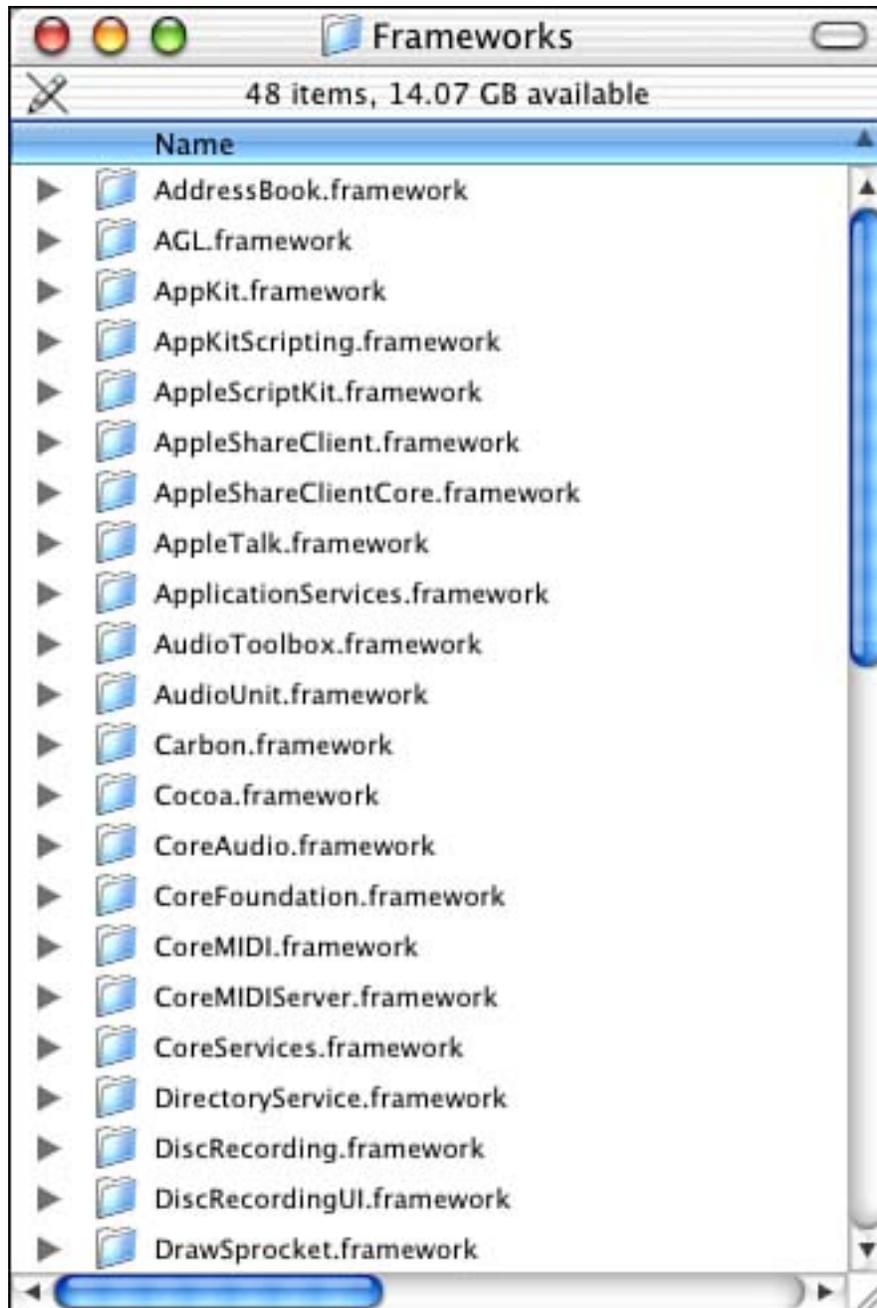
◀ PREVIOUS

NEXT ▶

Apple Frameworks

Much of the functionality in Mac OS X is implemented via frameworks. The AudioUnits framework is one such framework that implements audio-related functionality in the OS. There are plenty of others as well. Browse the `/System/Library/Frameworks` directory structure for a whole bunch of frameworks available for your use (see [Figure 6.2](#)). You can usually find documentation on Apple's Web site for each framework.

Figure 6.2. A partial look at the `/System/Library/Frameworks` directory.



As you can see, there are plenty of frameworks to keep you busy exploring and new ones are being made available regularly. Most of them are documented, but every once in awhile, you might find one that has no documentation—be careful with these. Sometimes Apple uses a framework to implement something

internally to the OS with out releasing the documentation right away, although the interface might exist. Eventually these interfaces usually become public, but if they are not officially documented, be wary of using them in any application you release to the public. You might find yourself with a headache later on when the interface changes.

Other frameworks exist in the `/System/Library/PrivateFrameworks` directory. These are truly private to Apple and can (and will) change at anytime without warning or concern for binary compatibility. Generally, anything in `/System/Library/Frameworks` is fair game for third-party developers such as yourself. Just because it isn't documented doesn't mean that it isn't supported. However, if you use anything in `/System/Library/PrivateFrameworks`, you can be guaranteed the headache that was only a possibility in the previous paragraph.

[\[Team LiB \]](#)

◀ PREVIOUS NEXT ▶

Third-Party Frameworks

Some very popular third-party frameworks are also available for your use. One in particular that is quite robust is made available by The Omni Group at <http://www.omnigroup.com/developer/sourcecode/>. The Omni Group provides multiple frameworks that provide varying functionality. All is available for your use under the Omni Source License (available on their Web site), and some of it is even cross-platform. Let's look at what it has to offer as of this writing. The following descriptions are directly from the Web site with only a few edits.

OmniBase

OmniBase is the lowest-level framework in the Omni framework suite. It is used by virtually every consumer product and consulting application we've written. OmniBase provides a series of debugging aids for class allocation and initialization, an alternative assertions mechanism, several Objective-C runtime manipulation aids, and a very reliable, cross-platform implementation of `+load:` (called `+didLoad:`).

OmniFoundation

OmniFoundation is our extension to Apple's Foundation framework. Beside several extremely useful extensions to Apple Foundation classes, OmniFoundation provides a horde of unique and powerful classes. Among the more interesting are

- `OFStringScanner`— A blindingly fast Unicode-safe alternative to `NSStringScanner` for when you really need to burn through those character streams.
- `OFRegularExpression`— Powerful regular expression processing wrapped in an Objective-C shell. Combine with `OFStringScanner` for extra fun.
- `OF Trie`— Implementation of the popular trie data structure. Interfaces to `OFStringScanner` for rapid scanning of "longest" token elements.
- `OFMessageQueue` and `OFQueueProcessor`— Writing stable multithreaded applications is an error prone process. These classes help de-fang the savage beast.

OmniFoundation is a veritable treasure chest of programming goodies. Look for yourself.

OmniNetworking

OmniNetworking provides a simple and extensible Objective-C wrapper to a multitude of complex networking constructs. Communication over several Internet standard protocols is supported, including TCP, UDP, and Multicast. Writing a simple FTP client or custom TCP/IP-based server program becomes a trivial task using OmniNetworking.

OmniAppKit

OmniAppKit is our set of extensions to Apple's AppKit.framework—full of cool stuff to make Mac OS X application development even easier. Some highlights are

- AppleScript— `OAOSAScript` makes it super easy to run AppleScripts from within a Cocoa app, and

companion classes introduce features such as automatically populated script menus, scripts on toolbars, and extensions to the Text suite allowing scripters to manipulate rich text.

- **OAPreferences**— An architecture for building multipane Preferences windows. All you need to do is write the individual preference panes (which is quite easy) and **OAPreferences** will automatically generate an appropriate Preferences window—one such as System Preferences if you have a lot of panes, or one such as Mail's Preferences window if you only have a few.
- **OAFindPanel** — Apple has a standardized architecture for Find in applications, but implementing it is entirely up to developers. OmniAppKit provides a powerful Find and Replace panel and a protocol for easily making widgets and documents in your app searchable.

OmniAppKit also includes a lot of extensions to Apple's classes to make creating rich user interfaces much easier.

OWF

Perhaps the most powerful framework in our entire framework suite, OWF, otherwise known as the Omni Web Framework, provides an advanced architecture within which to write multithreaded, Internet applications. OWF is the work horse in OmniWeb controlling all the content fetching, HTML/SGML parsing, FTP server manipulation, and so on. If OmniFoundation is the Objective-C programmer's Swiss Army Knife, OWF is their double barrel, rotary laser cannon.

Want to write a quick app to fetch stock prices from www.fakestocksite.com? OWF practically already does it. Need to fetch and parse a credit report from the credit bureau in your e-commerce site? Sprinkle in some custom logic, and the OWF fairy does the rest. In the next version, we plan on integrating back scratching and toast making.

OmniHTML

OmniHTML is the part of OmniWeb's architecture that actually renders parsed HTML and decoded images for display in a view. It also provides some conveniences such as URL dragging and file downloading that come in handy if you're writing something such as a Web browser.

As you can see, The Omni Group makes some powerful frameworks available. Be sure to check them out if you need any of the functionality they provide. This is proven code that you wouldn't want to rewrite if you didn't have to.

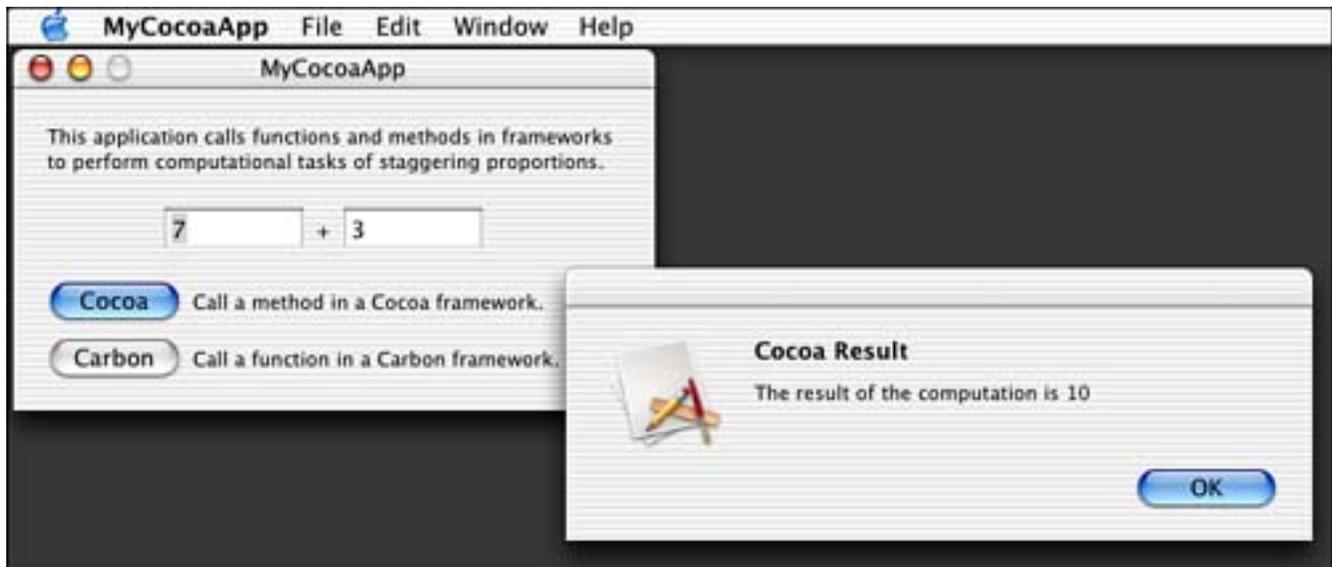
Now let's look at how to create our own framework!

[\[Team LiB \]](#)

The Result

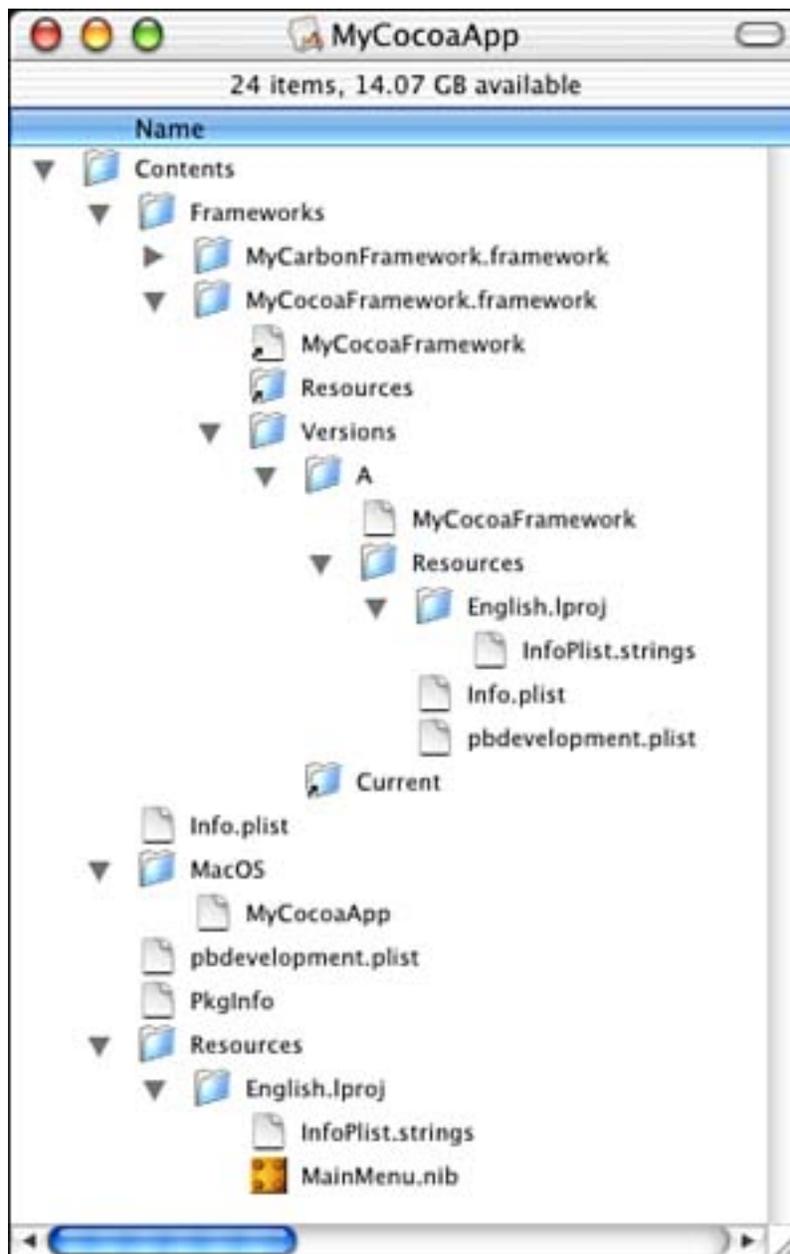
[Figure 6.3](#) shows the resulting application in action. Our application is written in Cocoa and calls through to both a Carbon and a Cocoa framework, depending on which button you press. The two integer values from the fields above the buttons are passed to the framework, and the result is returned and displayed in the alert. The application handles displaying all user interface components including the alert. In our case, the framework has no user interface at all and simply performs the calculation.

Figure 6.3. MyCocoaApp calling a framework.



[Figure 6.4](#) shows what the Cocoa application's package looks like. Note that there is a Frameworks directory containing both the Carbon and Cocoa frameworks. Much like how plug-ins are copied into an application, we copy the frameworks into our application here. This is discussed in more detail later. We could have placed the frameworks in the `~/Library/Application Support/Our Application` directory if we rather, but in this case it's easier to build them right into the application itself—no installer required! Mind you, by putting the frameworks directly in the application, they cannot be shared by other applications that might also use them. This is our trade-off at this point. `/Library/Frameworks` is also a popular place to store frameworks that need to be shared among multiple applications.

Figure 6.4. MyCocoaApp package contents.



Note

How do you decide if you should put a framework inside your application package or in a friendlier place to allow sharing between other applications? If your framework is private and no other applications (including your own) will use it, build it in to your application package. However, if you are working with a framework that will be used in multiple applications, save your users some drive space (and memory) and place them in a universal location to be shared among all the applications that will use them.

Taking this one step further, frameworks can be made available as application-specific (as noted previously), user-specific, machine-wide or across an entire local area network. Placing your framework in `~/Library/Frameworks` makes it available to only that specific user. Placing your framework in `/Library/Frameworks` makes it available to all users on the machine. Finally, if you want to use your framework over an entire network, you would place it in `/Network/Library/Frameworks`.

In this chapter, we will create three projects. One is the Cocoa application that will call the frameworks and the other two are the frameworks themselves. One of the frameworks will be implemented as a Carbon framework; the other will be a Cocoa framework. Let's look at the projects!

[\[Team LIB \]](#)

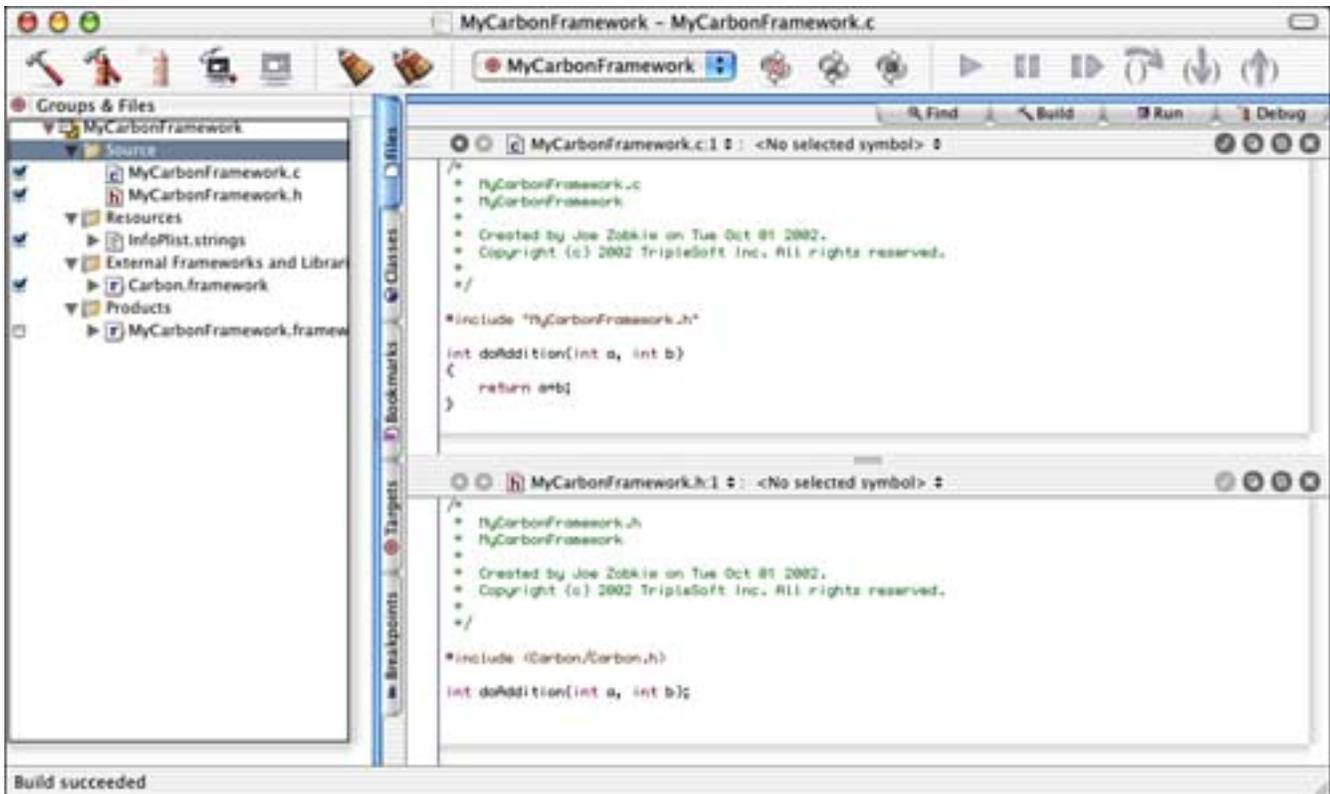
◀ PREVIOUS

NEXT ▶

The Carbon Framework Project

This project (see [Figure 6.5](#)) was started with the Carbon Framework template in Project Builder. Other than creating the `MyCarbonFramework.c` and `MyCarbonFramework.h` files as shown, there really is nothing more to do to have a working Carbon framework. Simply build the project, and `MyCarbonFramework.framework` will be created for you.

Figure 6.5. The MyCarbonFramework project in Project Builder.



[Listings 6.2](#) and [6.3](#) show that this framework has one function, `doAddition`. It takes two integer parameters, `a` and `b`, and returns a single integer result. We could have added as many functions and source files as needed to this project. We could even add resources and nibs. As long as it all compiles, we're all set. Let's look at the Cocoa equivalent, and then we'll see how the Cocoa application makes use of these frameworks.

Listing 6.2 `MyCarbonFramework.h`

```
#include <Carbon/Carbon.h>

int doAddition(int a, int b);
```

Listing 6.3 `MyCarbonFramework.c`

```
#include "MyCarbonFramework.h"
```

```
int doAddition(int a, int b)
{
    return a+b;
}
```

[\[Team LiB \]](#)

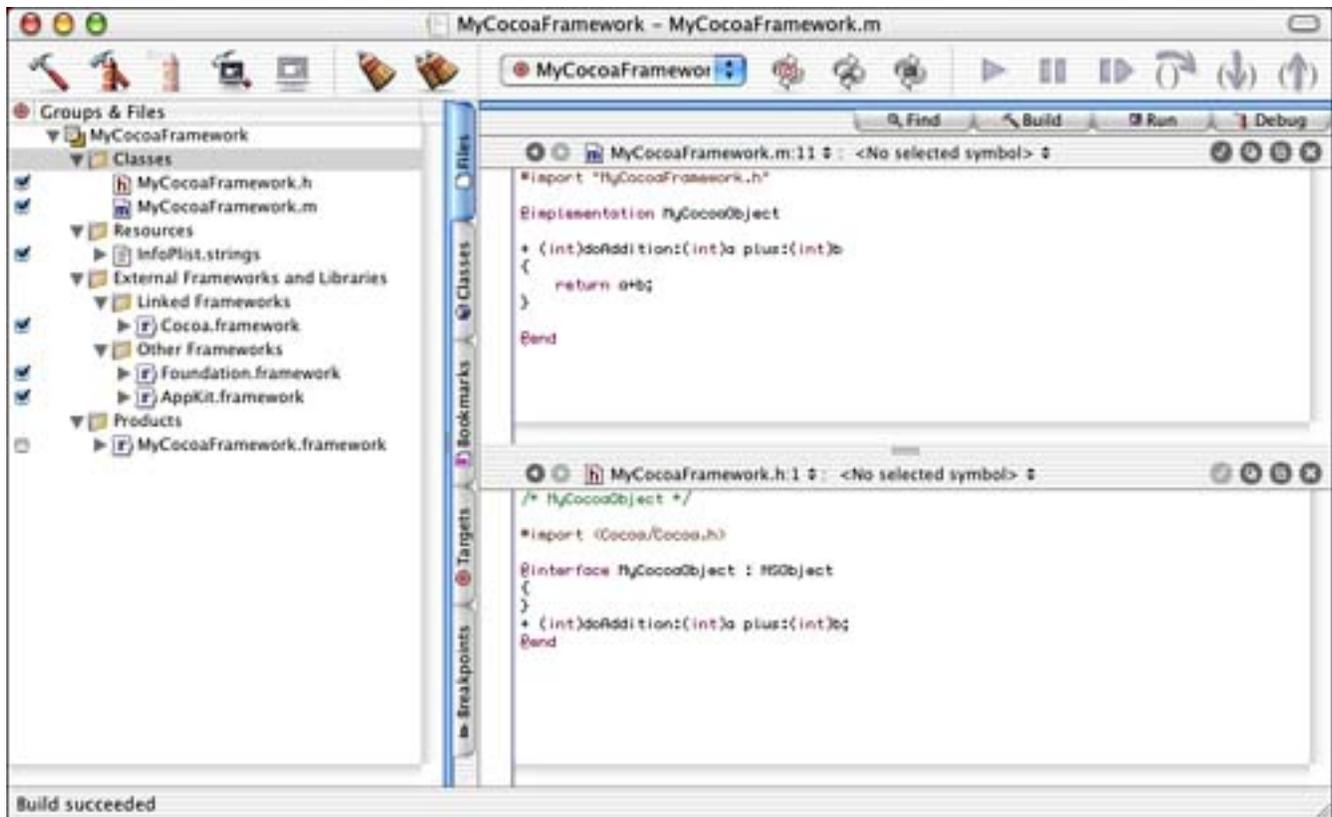
◀ PREVIOUS

NEXT ▶

The Cocoa Framework Project

This project (see [Figure 6.6](#)) was started with the Cocoa Framework template in Project Builder. Other than creating the `MyCocoaFramework.m` and `MyCocoaFramework.h` files as shown, there really is nothing more to do to have a working Cocoa framework. Simply build the project, and `MyCocoaFramework.framework` will be created for you.

Figure 6.6. The MyCocoaFramework project in Project Builder.



[Listings 6.4](#) and [6.5](#) show that this framework has one class that has a single class method, `+doAddition:plus:`. It takes two integer parameters, `a` and `b`, and returns a single integer result. We could have added as many classes, methods, and source files as needed to this project. We could even add resources and nibs. As long as it all compiles, we're all set. See how similar the Carbon and Cocoa frameworks are? They both accomplish the same task with only slight differences. On disk, they both pretty much look the same as well.

Listing 6.4 MyCocoaFramework.h

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

@interface MyCocoaObject : NSObject
{
}
+ (int)doAddition:(int)a plus:(int)b;
@end
```

Listing 6.5 MyCocoaFramework.m

```
#import "MyCocoaFramework.h"

@implementation MyCocoaObject

+ (int)doAddition: (int) a plus: (int) b
{
    return a+b;
}

@end
```

Note

Question: Why did we make this a class method?

Answer: Because no object state was required to satisfy the result.

This method could have easily been implemented as a C-style function as in the Carbon example. However, I wanted to show a method as part of an object. Any framework you implement might be more complex than this example, and you might need to implement class and instance methods.

[\[Team LiB \]](#)

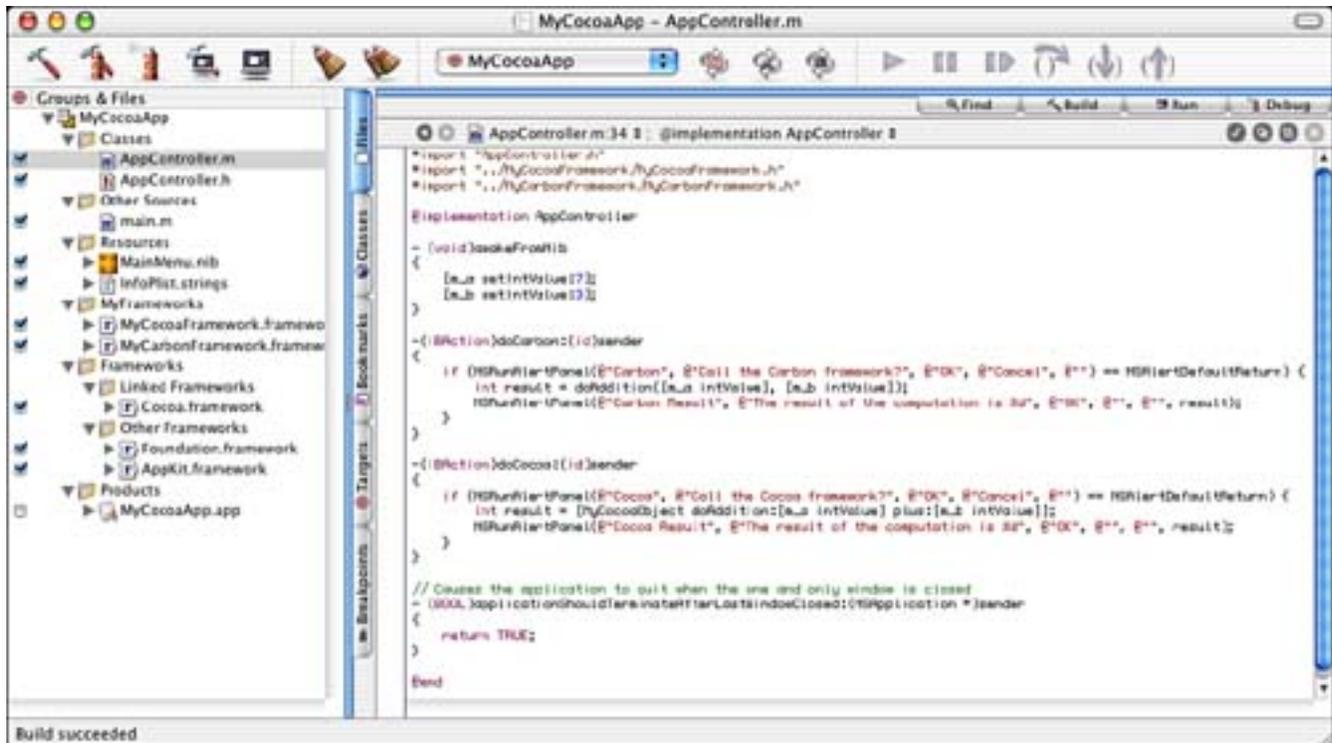
◀ PREVIOUS

NEXT ▶

The Cocoa Application Project

Our Cocoa application is straightforward as well. We created a basic application project (see [Figure 6.7](#)) using our `AppController` to both implement the `-applicationShouldTerminateAfterLastWindowClosed:` and function as our window controller by managing the `NSTextField`s in the application window and the `-doCarbon:` and `-doCocoa:` action methods.

Figure 6.7. The MyCocoaApp project in Project Builder.



Note

Remember, in order for the `AppController` to receive the `-applicationShouldTerminateAfterLastWindowClosed:` delegate message, it must be registered as the application's delegate. One way to do this is to hook your `AppController` object up to the File's Owner's (`NSApplication`'s) delegate outlet in `MainMenu.nib`. Another way is to send `[NSApp setDelegate:self];` in `AppController`'s `-awakeFromNib` method.

In `AppController.h`, the `AppController` object is defined as shown in [Listing 6.6](#). Note that it has two `IBOutlet`s for the text fields and two `IBAction`s for the buttons. These are connected in the standard way using Interface Builder.

Listing 6.6 `AppController` Interface in `AppController.h`

```

@interface ApplicationController : NSObject
{
    IBOutlet NSTextField *_m_a;
    IBOutlet NSTextField *_m_b;
}
- (IBAction) doCarbon: (id) sender;
- (IBAction) doCocoa: (id) sender;
@end

```

[Listing 6.7](#) shows the `AppController` implementation itself. Notice that we import the header files from the two frameworks in order to have access to the functions/methods within them. In addition, at the top of this file, in `awakeFromNib`: we initialize the values of the two `NSTextField`s to 7 and 3. They are numbers chosen arbitrarily—they could have just as easily been 39 and 43. Jumping to the bottom of the file, the `applicationShouldTerminateAfterLastWindowClosed`: method is implemented as we've done in the past, for convenience.

The `-doCarbon:` and `-doCocoa:` methods are the meat of this example. When the user clicks the Carbon button, after giving her the chance to cancel, the `-doCarbon:` method calls the `doAddition` function in the Carbon framework. It passes the integer values from the text fields and displays the result in an alert. The `-doCocoa:` method works the same way with the exception that the framework that it calls contains a Cocoa class with `+doAddition:plus:` implemented as a class method.

Listing 6.7 `AppController` Implementation in `AppController.m`

```

#import "AppController.h"
#import "../MyCocoaFramework/MyCocoaFramework.h"
#import "../MyCarbonFramework/MyCarbonFramework.h"

@implementation ApplicationController

- (void) awakeFromNib
{
    [_m_a setIntValue: 7];
    [_m_b setIntValue: 3];
}

- (IBAction) doCarbon: (id) sender
{
    if (NSRunAlertPanel(@"Carbon",
        @"Call the Carbon framework?", @"OK", @"Cancel", @"")
        == NSAlertDefaultReturn) {
        int result = doAddition([_m_a intValue], [_m_b intValue]);
        NSRunAlertPanel(@"Carbon Result",
            @"The result of the computation is %d",
            @"OK", @"", @"", result);
    }
}

- (IBAction) doCocoa: (id) sender
{
    if (NSRunAlertPanel(@"Cocoa",
        @"Call the Cocoa framework?", @"OK", @"Cancel", @"")
        == NSAlertDefaultReturn) {
        int result = [MyCocoaObject doAddition:[_m_a intValue]

```

```

        plus: [m_b intValue]);
NSRunAlertPanel(@"Cocoa Result",
                @"The result of the computation is %d",
                @"OK", @"", @"", result);
    }
}

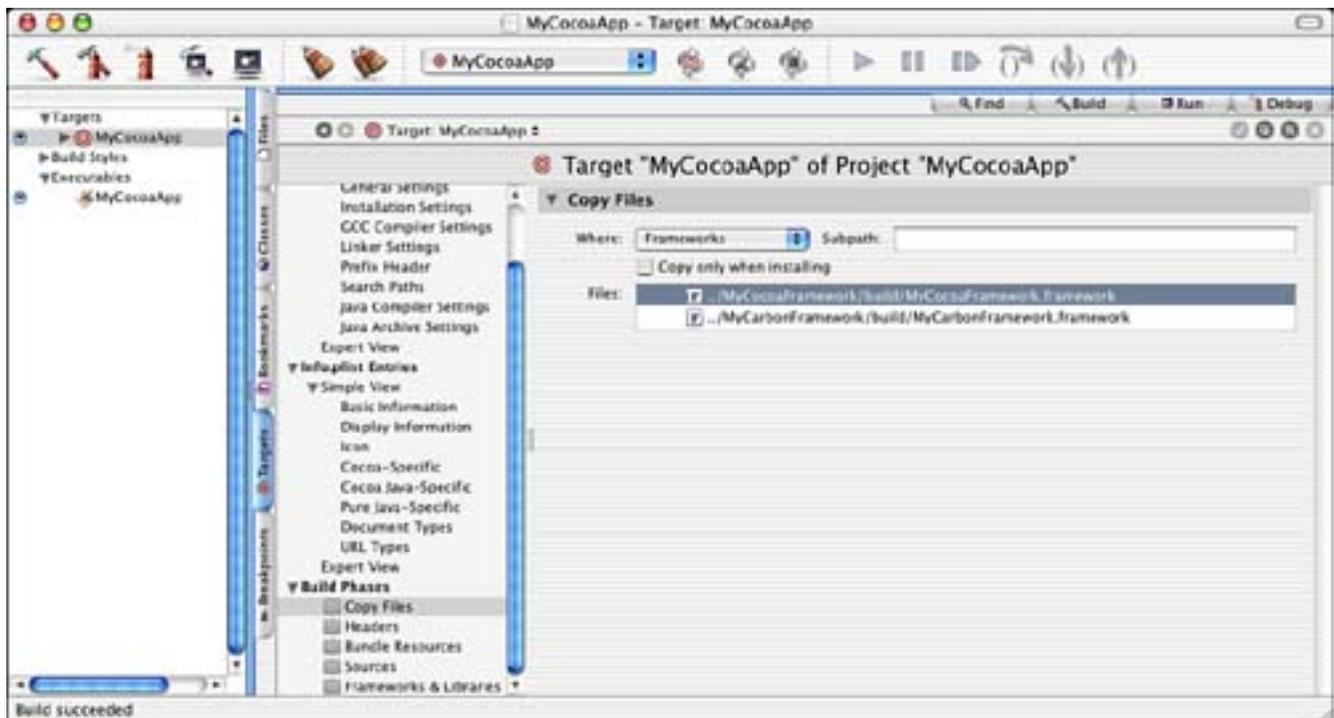
// Causes the application to quit when the one and only window is closed
- (BOOL) applicationShouldTerminateAfterLastWindowClosed: (NSApplication *) sender
{
    return TRUE;
}

@end

```

One last thing to remember is that the frameworks themselves must be copied into the application bundle. [Figure 6.8](#) shows the Copy Files Build Phase, as you've seen used in previous chapters. Note that we are copying the files to the Frameworks directory as opposed to the Plug-Ins directory that we've used in the past. If you forget this step, your application will simply quit as soon as it launches. You might see a few bounces in the dock, but then it will fade away.

Figure 6.8. MyCocoaApp Copy Files build phase in Project Builder.



That is how you call a Cocoa or Carbon framework from within a Cocoa application.

Try This

Here are some ideas to expand the framework project in this chapter; try these on your own.

Add another method to the Cocoa framework, but make it an instance method instead of a class method. How would you call this method?

Create a **doSubtraction** implementation to each framework. Change the user interface to support addition or subtraction of the same numbers. You might consider using a pop-up button or radio buttons to allow the user to choose.

Instead of putting the frameworks in the application package itself, place them in a location that can be used to share the frameworks with other applications.

Conclusion

As you can see, frameworks are a great way to manage your source code. Even if you don't need to hide the implementation from others, packaging your tested modules as frameworks has many advantages for code reuse and distribution—even within your organization. Think about how you might factor your application to make it leverage the power of frameworks.

Part III: Enhancing the System

[7 System Services](#)

[8 System Preference Panes](#)

[9 Status Items](#)

[10 Screen Effects](#)

[11 Color Pickers](#)

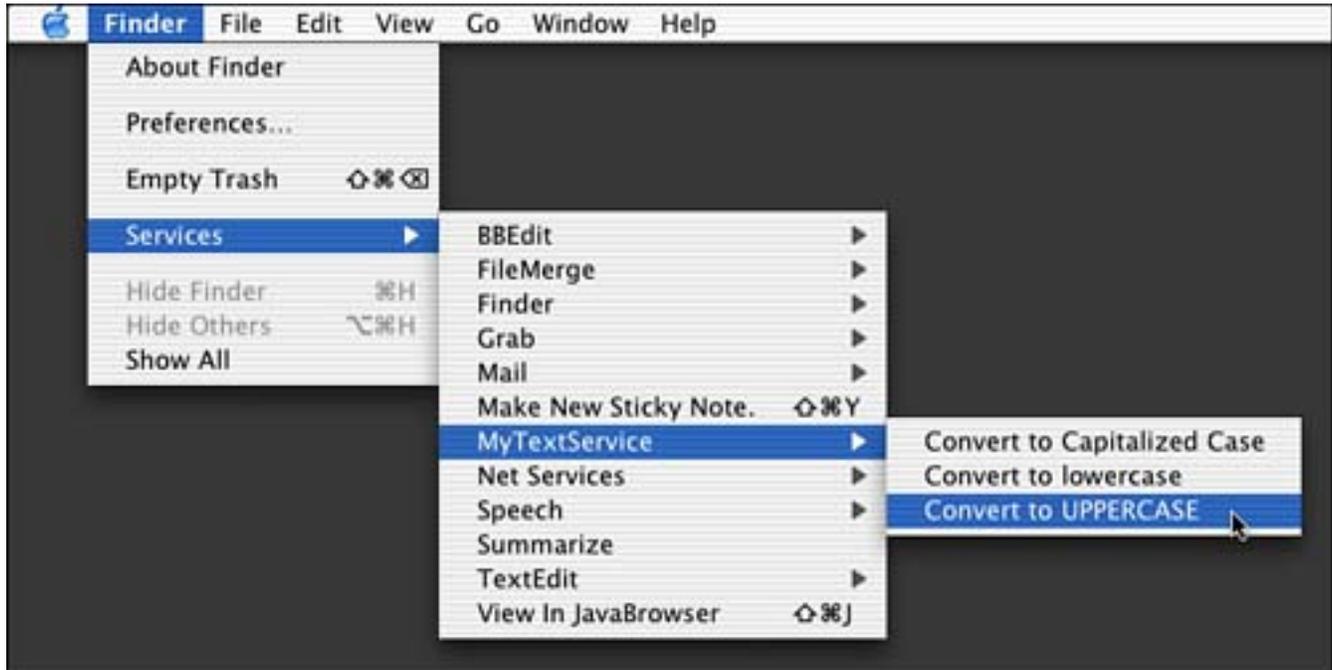
Chapter 7. System Services

"Service with a smile!"

—Business slogan

System Services, hereinafter referred to as services, are applications that provide custom capabilities to other applications. They can be found in the Services submenu of the application menu, as shown in [Figure 7.1](#). Services are usually background-only applications and do not have a user interface as a normal application would, although they might display a window for user input or feedback if needed. Examples of services include spell checkers, screen grabbers, and text manipulators such as the one we will look at in this chapter.

Figure 7.1. The Services menu.



Services are installed in either the `Applications` or `/Library/Services` folders. Those in the `Applications` folder are normal applications and must be run once in order to have their items added to the Services menu. They are then "remembered" by the operating system. Those in the `/Library/Services` folder are loaded at login.

The items added to the menu can either be commands or submenus of commands. In [Figure 7.1](#), the Summarize application offers a single service and is listed as a single menu item: Summarize. The MyTextService application, on the other hand, offers multiple services and therefore uses a submenu filled with commands.

Although the contents of the Services menu are populated when an application launches, the items are disabled by default. When the user clicks on the Services menu, the responder chain in the current application is traversed to locate objects that can read or write the types of data provided by the services in

the menu. If any are found, those items are enabled in the menu. For example, you wouldn't have text manipulation services enabled if you currently had an image selected in a window. Many Cocoa objects provide default behaviors for the various clipboard types. `NSTextField`, for example, can automatically handle text-based services. See the later section "[The Application Connection](#)" for more information on how your application can make use of specific services.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Result

In this chapter, I will discuss a service called MyTextService, as shown in [Figure 7.1](#). MyTextService provides three standard manipulations that can be performed on any selected text: uppercase, lowercase, or capitalized case. Selected text will be edited in place. [Figures 7.2](#) and [7.3](#) show the TextEdit application before and after selecting Convert to Capitalized Case. Note how the lowercase and uppercase text is converted immediately, in place, to capitalized case.

Figure 7.2. TextEdit just before selecting Convert to Capitalized Case from the MyTextService menu.

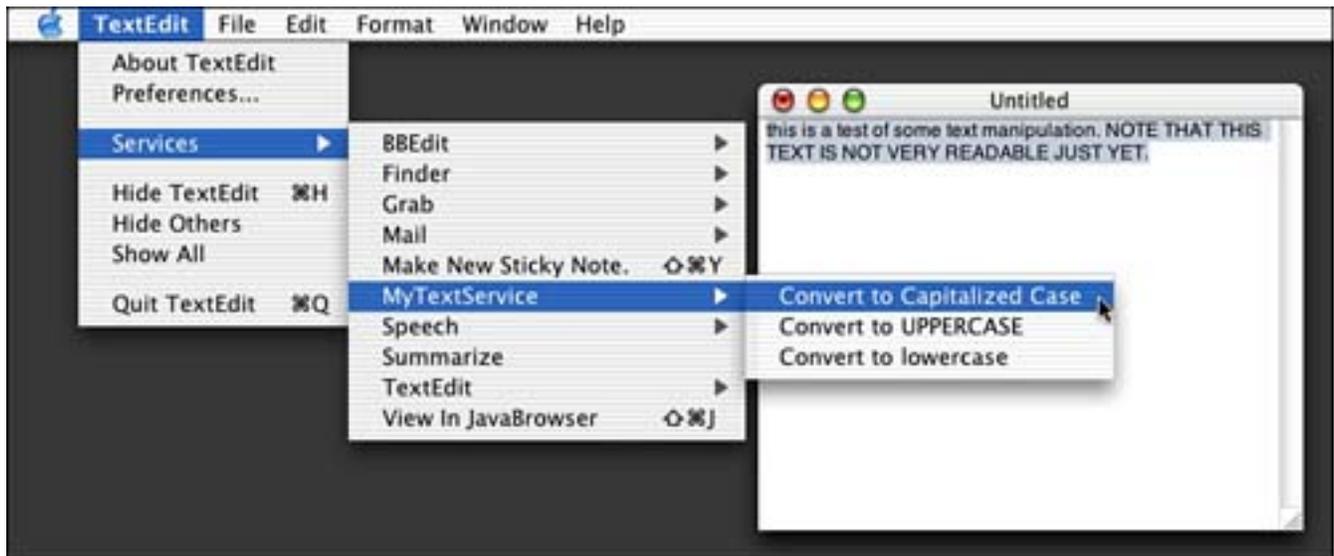
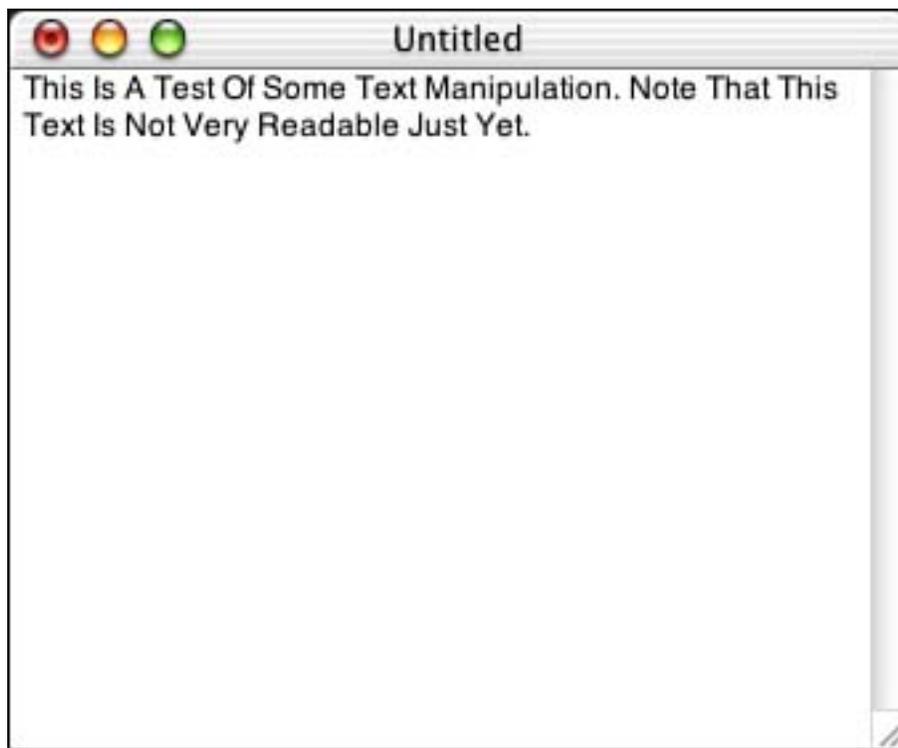


Figure 7.3. TextEdit just after selecting Convert to Capitalized Case from the MyTextService menu.



Let's look at the project.

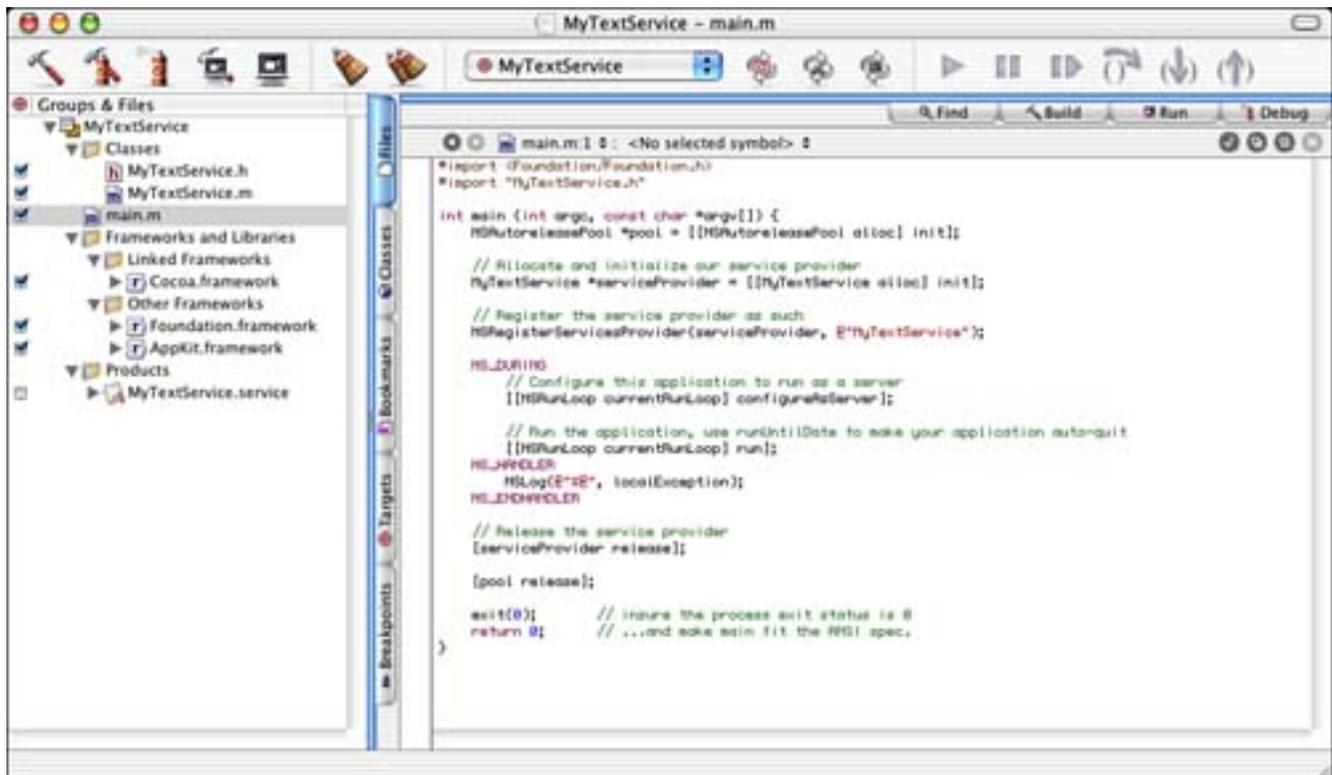
[\[Team LiB \]](#)

◀ PREVIOUS NEXT ▶

The Project

[Figure 7.4](#) shows the MyTextService project in Project Builder. You can create this project using the Cocoa Bundle template in the New Project Assistant, although you will need to make some changes, as discussed next. Let's look at the project settings.

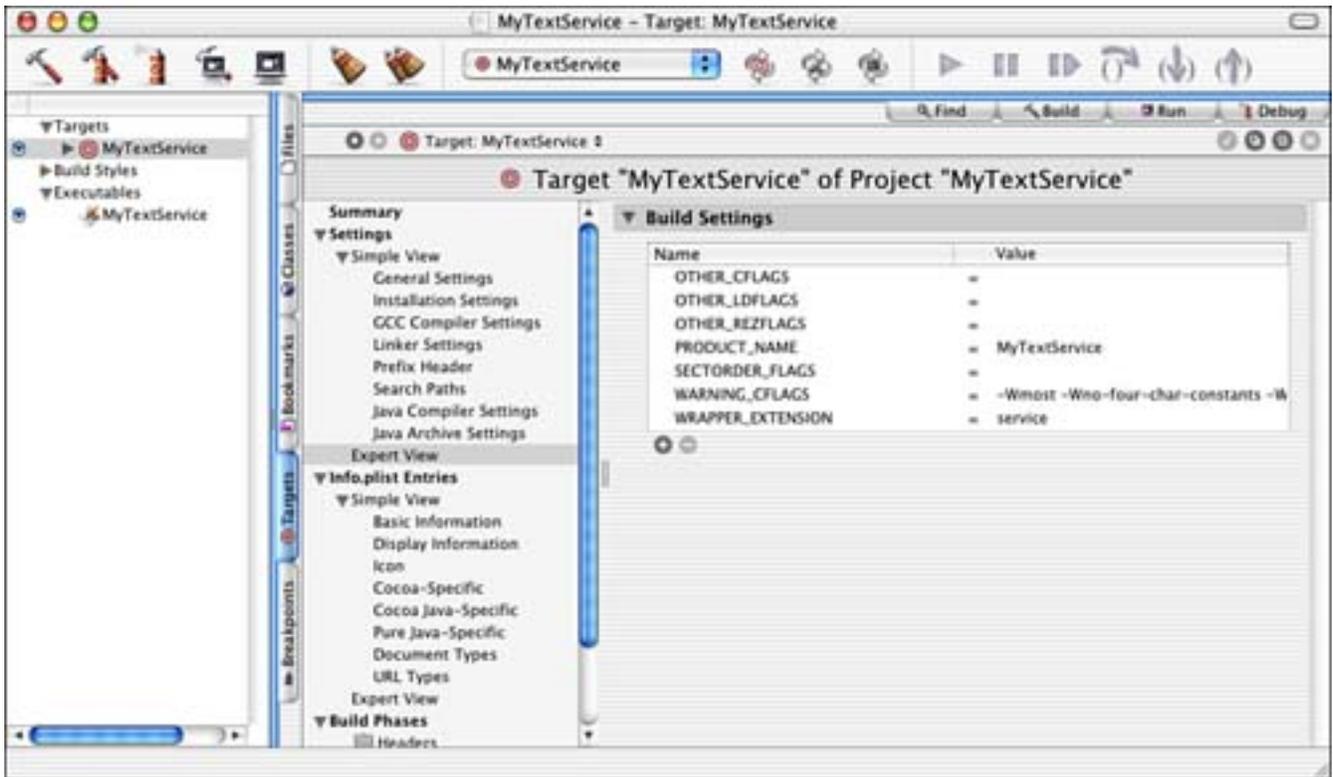
Figure 7.4. The MyTextService project.



The Project Settings

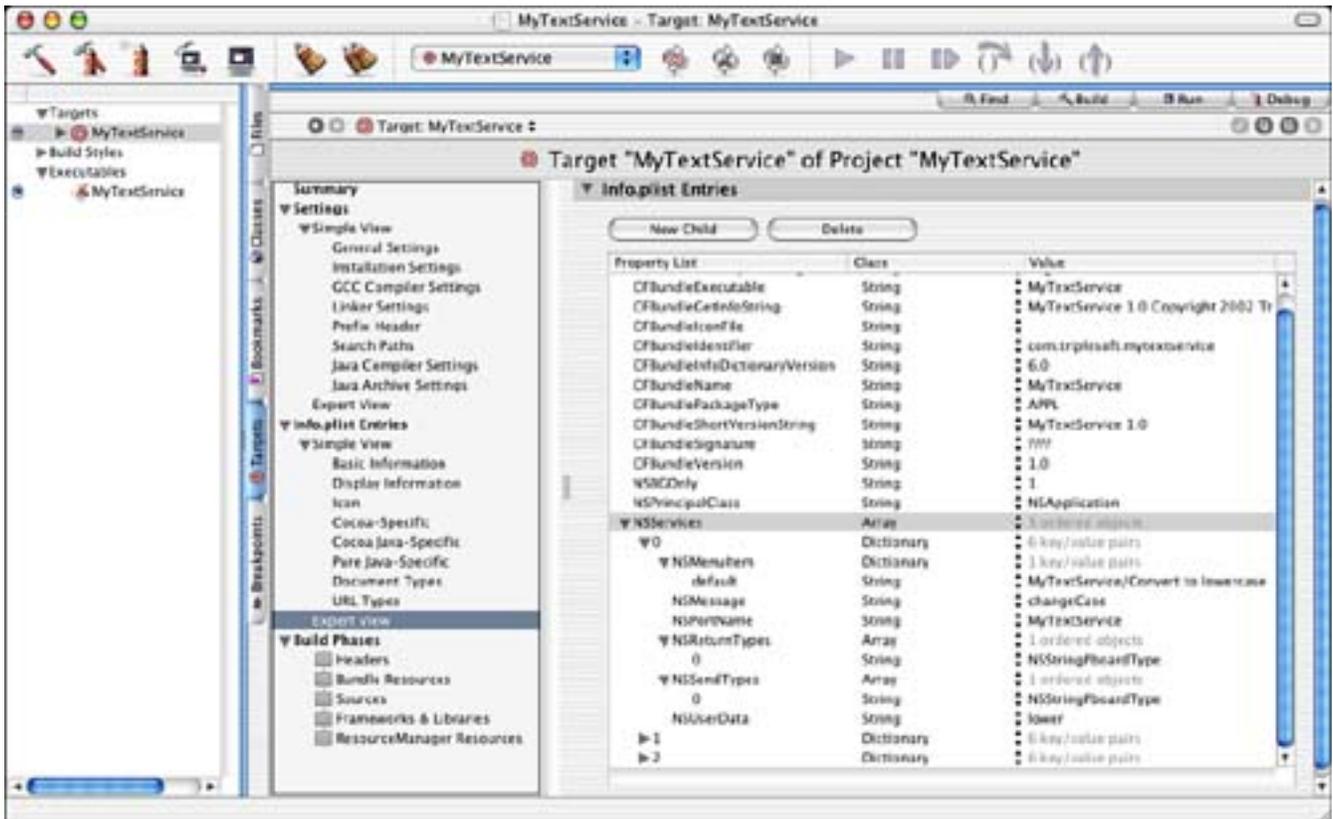
The first thing you will want to verify is that in the Target settings, the `WRAPPER_EXTENSION` is set to `service`. As mentioned previously, applications can provide services as well, but we will be implementing a background-only version of our service that will live in the `/Library/Services` folder. If we do not use the service extension, the operating system will not recognize and load our service properly (see [Figure 7.5](#)).

Figure 7.5. The MyTextService Target settings.



The next project settings to be concerned with are much more involved—the InfoPlist settings, as shown in [Figure 7.6](#). Many of these items will look familiar, but there are a few distinct additions and differences for this project than you've seen previously.

Figure 7.6. The MyTextService InfoPlist entries.



First, you will want to make sure that the `CFBundlePackageType` entry is set to `APPL` and not `BNDL`. Although we used the Bundle template project, we are really writing an application. You will also want to set the `NSPrincipalClass` entry to `NSApplication`. The last application-specific item is the `NSBackgroundOnly` entry. You will need to add this entry yourself if it does not exist in your template and set its value to `1`. This tells the operating system that this application has no user interface and will therefore not appear in the Dock. This is key to creating what can be called a faceless background application—an application that runs silently, performing its duty unseen by the user.

Note

As you've learned in the past, there is more than one way to skin the Cocoa cat. You might consider starting with an application template as opposed to a Bundle template. Either way, you will have to tweak a few items in the project settings.

The next entry you will see is the `NSServices` array. This array is the heart and soul of the Service mechanism. It contains an array of dictionaries that specify the services offered by `MyTextService`. I will discuss item 0 in this array, but items 1 and 2 are similar with the exception of the menu item name and user data that they pass to `MyTextService`.

The first thing you see in array item 0 is a dictionary item named `NSMenuItem`. This dictionary contains a solitary item named `default`, which specifies the name of the menu item to be added to the Services menu. You can specify a submenu name as well by using a `/` in this string. Because `MyTextService` provides multiple services, we add a submenu named `MyTextService` and then the specific text of the service to be offered; in this case, `Convert to lowercase`.

You can optionally add a keyboard equivalent for your menu items by adding a `NSKeyEquivalent` dictionary to the plist. Similar to the `NSMenuItem` dictionary, the `NSKeyEquivalent` dictionary consists of a single item named `default`. You should use keyboard equivalents sparingly because the Services menu is a systemwide feature and you could easily use a key combination already in use elsewhere in the system or application you are trying to serve. My suggestion is to avoid them unless necessary.

Next is the `NSMessage` entry. This entry is a string that represents the method name to be invoked in the application's service provider object. The operating system will construct an Objective-C method of the form `-methodName:userData:error:` and call the method in that manner. You will see this method implemented later in this chapter.

`NSPortName` is the port to which the application should listen for service requests. In most cases, this is the application name itself. Therefore, in our case, it is `MyTextService`.

`NSReturnTypes` and `NSSendTypes` are two array entries that list the types of data that the service is capable of reading and writing. In this example, we only deal with text data, so we list `NSStringPboardType` in both entries. You can add as many types to these arrays as required. These values are used when the operating system looks for matches between the types of data an application has to offer and what services can manipulate it.

Note

`NSPasteboard.h` defines many of the standard pasteboard types. These include Strings,

Filenames, PostScript, TIFF, RTF, RTFD, Tabular Text, Font Styles, Rulers, File Contents, Colors, HTML, PICT, URL, PDF, Vcard, and more.

`NSUserDefaults` is a string value that is passed in the `userData` parameter of the `NSMessage`-defined method call. Although you do not have to make use of this field, I chose to implement `MyTextService` by using the same `NSMessage` entry for all three of our services and using the `NSUserDefaults` entry to specify which formatting to apply to the incoming text: lowercase, uppercase, or capitalization. We could have just as easily had three different `NSMessage` entries—one for each formatting option.

Last, the optional `NSTimeInterval` numerical string entry can be used to indicate the number of milliseconds that the operating system should wait for a response from a service. This only applies when a response is required. By default, if it is not specified, the `NSTimeInterval` is 30 seconds (30,000 milliseconds). If the `NSTimeInterval` value is exceeded, the operating system aborts the service request and continues.

As you can see, you have a lot of control over the specifics of your service. Let's see what the source code looks like that brings this all together.

The Source Code

There are only a few listings to be concerned about in this project. The application itself needs a `main` function, but it's different from what you normally have seen. In fact, it's about 20 times the size of a normal Cocoa application's `main` function (see [Listing 7.1](#)).

Listing 7.1 `MyTextService main`

```
#import <Foundation/Foundation.h>
#import "MyTextService.h"

int main (int argc, const char *argv[]) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    // Allocate and initialize our service provider
    MyTextService *serviceProvider = [[MyTextService alloc] init];

    // Register the service provider as such
    NSRegisterServicesProvider(serviceProvider, @"MyTextService");

    NS_DURING
    // Configure this application to run as a server
    [[NSRunLoop currentRunLoop] configureAsServer];

    // Run the application, use runUntilDate to make your
    // application auto-quit
    [[NSRunLoop currentRunLoop] run];
    NS_HANDLER
    NSLog(@"%@", [localException]);
    NS_ENDHANDLER

    // Release the service provider
    [serviceProvider release];
}
```

```

    [pool release];

    exit(0);          // insure the process exit status is 0
    return 0;        // ...and make main fit the ANSI spec.
}

```

We start out by allocating and initializing our own `NSAutoReleasePool` to take care of any autorelease memory used throughout the running of our application. We then allocate and initialize our `MyTextService` object. Once allocated, we call `NSRegisterServicesProvider` to register our provider by name with the operating system's services mechanism. This is where the `NSPortName` Info.plist entry comes into play—these strings should match exactly.

Next, you will notice a block of code that includes the macros `NS_DURING`, `NS_HANDLER`, and `NS_ENDHANDLER`. These macros are defined in `NSException.h` and are similar to the `TRY/CATCH` block you might be familiar with from the C++ programming language. This is our safety net should any exceptions occur during the execution of the application. The `NSLog` function will display the exception information.

We then obtain the current `NSRunLoop` instance of the current thread (actually creating a new one) by calling the `NSRunLoop` class method `+currentRunLoop`. Calling `-configureAsServer` makes the `NSRunLoop` instance suitable for use by a server process—currently a no-op under Mac OS X, but included for historical consistency. Calling the `-run` method causes the application to begin running.

Note

`NSRunLoop`'s `-runUntilDate:` method could have been called instead of the `-run` method. This allows us to autoquit our application after a certain amount of time. By calling `-run`, our service will execute until the machine shuts down, restarts, or the user logs out.

Having said that, some programmers, including one named Mike, would consider this current behavior a bug. Services that do not need to perform some type of background processing (network IO, data processing, and so on) should quit as soon as they are done performing their service. This would be especially true if the service consumed large amounts of memory while it was idle. There's no sense in hanging around when you might no longer be needed.

Once the application has exited, we can `-release` our service provider object, `-release` the autorelease pool, and `exit` the `main` function. You can also use the `applicationWillTerminate:` delegate message to handle any cleanup.

The `MyTextService` class is a subclass of `NSObject` with one method defined, `-changeCase:userData:error:`. This is where the `NSMessage` Info.plist entry comes into play—the name of this method and the value of `NSMessage` should match exactly (see [Listing 7.2](#)).

Listing 7.2 MyTextService Interface in MyTextService.h

```

@interface MyTextService : NSObject {
}

```

```
- (void) changeCase: (NSPasteboard *) pboard
    userData: (NSString *) userData
    error: (NSError **) error;
```

```
@end
```

The implementation of `MyTextService` does nothing more than implement the `changeCase:` method. Let's look at [Listing 7.3](#) in detail.

Listing 7.3 `MyTextService`—`changeCase:userData:error:` in `MyTextService.m`

```
@implementation MyTextService
```

```
- (void) changeCase: (NSPasteboard *) pboard
    userData: (NSString *) userData
    error: (NSError **) error
{
    NSString *pboardString;
    NSString *newString;
    NSArray *types;
    Boolean success;

    // Verify the types currently on the pasteboard
    types = [pboard types];
    if (![types containsObject: NSStringPboardType]) {
        *error = NSLocalizedString(@"Error: couldn't edit text.",
                                   @"pboard doesn't have a string.");
        return;
    }
    pboardString = [pboard stringForType: NSStringPboardType];
    if (!pboardString) {
        *error = NSLocalizedString(@"Error: couldn't edit text.",
                                   @"pboard couldn't give string.");
        return;
    }
    // Compare the mode so we do the correct thing
    if ([userData isEqualToString:@"upper"]) {
        newString = [pboardString uppercaseString];
    } else if ([userData isEqualToString:@"lower"]) {
        newString = [pboardString lowercaseString];
    } else if ([userData isEqualToString:@"cap"]) {
        newString = [pboardString capitalizedString];
    }
    if (!newString) {
        *error = NSLocalizedString(@"Error: couldn't edit text.",
                                   @"pboardString couldn't edit letters.");
        return;
    }
    types = [NSArray arrayWithObject: NSStringPboardType];

    // Load the value of the checkbox from the Preference Pane example
    success = CFPreferencesAppSynchronize(
        CFSTR("com.triplesoft.mypreferencepane"));
    // Force a synchronization so we get the latest "live" preference
    if (success && CFPreferencesGetAppBooleanValue(
```

```

        CFSTR("Boolean Value Key"),
        CFSTR("com.triplesoft.mypreferencepane"), NULL)) {
// Use [NSPasteboard generalPasteboard] instead of pboard to
// send the results to the clipboard
[[NSPasteboard generalPasteboard] declareTypes:types owner:nil];
[[NSPasteboard generalPasteboard] setString:newString
    forState:NSStringPboardType];
} else {
    // Send the results directly back to the app who requested it
    // if set or if key does not exist
    [pboard declareTypes:types owner:nil];
    [pboard setString:newString forState:NSStringPboardType];
}

return;
}

@end

```

This method takes three parameters: the `NSPasteboard` that contains the data in a format that the service can process, the `userData NSString` from the `NSUserDefaults` Info.plist entry, and an `error NSString`. The `NSPasteboard` can be edited in place, as you will see shortly. The human-readable `error NSString` can be returned should there be an error within our service.

The first thing we want to verify is that there are types on the `NSPasteboard` that we can handle. In theory, we should be covered here (remember our `NSReturnTypes` and `NSSendTypes` in Info.plist), but we check anyway. Assuming that there is an `NSStringPboardType`, we grab it from the `NSPasteboard`. We then compare the `userData NSString` and alter the text as appropriate using the `NSString` methods - `uppercaseString`, `lowercaseString`, or `capitalizedString`. If the conversion was successful, we then prepare to return it to the application that requested it.

You will notice that we look at the `com.triplesoft.mypreferencepane` preference settings using the `CFPreferencesAppSynchronize` and `CFPreferencesGetAppBooleanValue` functions. This is called foreshadowing. We look to a future chapter to see something that we will be adding as a feature. Actually, in [Chapter 8](#), "Preference Panes," we will implement a Preference Pane that will work with `MyTextService` to allow us to choose whether we should return the edited text directly back to the application that requested it or to the general (systemwide) pasteboard. Returning the text to the requesting application places the text directly back into the application, as shown in [Figures 7.2](#) and [7.3](#). Returning to the general pasteboard places the edited text in a buffer that allows you to paste it in any application you desire without altering the original data.

By default, the text will be pasted back into the requesting application, but the steps are the same no matter where we return it to. First, we create an `NSArray` of types, reusing the `types` variable, which contains one item representing the `NSStringPboardType`. We then call the `-declareTypes:owner:` method of the `NSPasteboard` in question passing the array. Last, we set the `newString` variable as type `NSStringPboardType` to the `NSPasteboard` by calling the `-setString:forType:` method. That's it!

Note

Note that in order for services to be loaded from the `/Library/Services` folder, you must log out and log in again after you first place it there. However, once it is registered with the operating system, you can replace it as many times as necessary during development, and your latest code will always be executed properly. Note, however, that the service is left

running once you select it from the Services menu. Therefore, you might have to quit it manually before you can replace the file for subsequent runs. Consider automatically quitting your daemonized service once it has performed its work to avoid this problem.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Application Connection

If you're writing a Cocoa application, most of the default user interface objects automatically implement and handle service menu functionality by default. That is, text-related objects know how to deal with the `NSStringPboardType`, and so on. However, if you are writing a custom class and you want that class to have the capability of using services, you can implement this mechanism in your class easily enough.

First, your custom class will need to be a subclass of `NSResponder` in order to respond to the events sent from the Services mechanism. In your custom class's `initialize` class method, you will need to call `NSApp`'s `+registerServicesMenuSendTypes:returnTypes:` class method passing the types that your object can send and return. These lists need not be the same. For example, you might accept a TIFF file and return an ASCII text formatted picture for use on the Internet. Your subclass can register any number of types, public or private. See [Listing 7.4](#) for an example of its use.

Listing 7.4 Registering Pasteboard Types

```
+ (void) initialize
{
    NSArray *sendTypes;
    NSArray *returnTypes;

    /* Make sure code only gets executed once. */
    static BOOL initialized = NO;
    if (initialized == YES) return;
    initialized = YES;

    sendTypes = [NSArray arrayWithObjects: NSStringPboardType, nil];
    returnTypes = [NSArray arrayWithObjects: NSStringPboardType, nil];
    [NSApp registerServicesMenuSendTypes: sendTypes returnTypes: returnTypes];
    return;
}
```

Your custom class will also receive calls to its `-validRequestorForSendType:returnType:` method throughout its instantiation. If your object can handle the type of data requested, you return `self`. Otherwise, you call through to the super class `-validRequestorForSendType:returnType:`. This method can be called many times during an event loop and so should be as fast as possible—there's no dillydallying in this method. See [Listing 7.5](#) for an example of its use.

Listing 7.5 Validating Pasteboard Types

```
- (id) validRequestorForSendType: (NSString *) sendType returnType: (NSString *) returnType
{
    if ((!sendType || [sendType isEqual: NSStringPboardType]) &&
        (!returnType || [returnType isEqual: NSStringPboardType])) {
        if ((!sendType || [self selection]) && (!returnType || [self isEditable]))
        {
            return self;
        }
    }
}
```

```
} return [super validRequestorForSendType: sendType returnType: returnType];  
}
```

Note

Be sure to read all about the `NSPasteboard` class. You can do many things by putting your data on the clipboard. Copying and pasting is one of those things that many developers try to avoid, but it can really add a kick to your application. It can make your application much more usable, especially if you allow data to be transferred with other applications. Just think if you couldn't paste an image into a Photoshop document! Where would we be today?

When the user selects an item from the Services menu, your custom class's `-writeSelectionToPasteboard:types:` method is called. This method must return the requested type of data to the service, returning `YES` if successful. The `-writeSelectionToPasteboard:types:` method is part of the `NSServicesRequests` protocol. See [Listing 7.6](#) for an example of its use.

Listing 7.6 Writing Pasteboard Types

```
- (BOOL)writeSelectionToPasteboard: (NSPasteboard *)pboard types: (NSArray *)types  
{  
    NSArray *typesDeclared;  
  
    if (![types containsObject: NSStringPboardType]) {  
        return NO;  
    }  
    typesDeclared = [NSArray arrayWithObject: NSStringPboardType];  
    [pboard declareTypes: typesDeclared owner: nil];  
    return [pboard setString: [self selection] ofType: NSStringPboardType];  
}
```

After the service has manipulated the data, it returns the data to the object by calling its `-readSelectionFromPasteboard:` method. The object then extracts the data from the pasteboard and displays it to the user. It should return `YES` if successful. The `-readSelectionFromPasteboard:` method is part of the `NSServicesRequests` protocol. See [Listing 7.7](#) for an example of its use.

Listing 7.7 Reading Pasteboard Types

```
- (BOOL)readSelectionFromPasteboard: (NSPasteboard *)pboard  
{  
    NSArray *types;  
    NSString *theText;  
  
    types = [pboard types];  
    if (![types containsObject: NSStringPboardType]) {  
        return NO;  
    }  
    theText = [pboard stringForType: NSStringPboardType];  
    [self replaceSelectionWithString: theText];  
    return YES;  
}
```

}

Note

Applications can create something called Add-on Services that are not directly in your application's bundle. You simply create a bundle with a `.service` extension that contains an `Info.plist` with the appropriate `NSServices` entries. The `NSMessage` and `NSPortName` values are those implemented in your application. This is a nice way to implement a plug-in style approach to services.

Here's one last thing of interest: Your application can invoke a service programmatically by calling the `NSPerformService` function. Although you normally would not use this, you can easily call a service by name, passing an `NSPasteboard` using this function. `NSPerformService` is defined in `NSApplication.h`.

That's about all there is to it to support Services in your application. In fact, if you only use the standard Cocoa types of user interface objects and don't need to support any custom data types on the clipboard, you don't even have to do this much!

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Try This

Here are some ideas to expand the text service project in this chapter; try these on your own.

Add some more advanced string manipulation functionality to the `-changeCase: userData: error:` method. You will add a new array item with different `NSMenuItem` and `NSUserDefaults` entries.

Add a method other than `-changeCase: userData: error:` to see how the service works with different `NSMessage` values.

Build a service that manipulates images instead of text. You can use some of the functionality shown in the Cocoa plug-ins described in an earlier chapter.

Conclusion

You learned a great deal in this chapter and explored your first operating system–level piece of code. We will continue this journey in subsequent chapters and see how to extend the operating system by adding little bits of specific code here and there. Many other projects are based on the Cocoa bundle project template, so stay tuned!

Chapter 8. System Preference Panes

"Use other door. It's broken."

—sign on donut shop door

System Preference Panes, hereinafter referred to as preference panes, are essentially plug-ins that are dynamically loaded by the System Preferences application. Preference panes and the System Preferences application allow users to alter systemwide settings using a standard host application. They live in the `/Library/PreferencePanes` folder in any of the standard search paths, but when you install new ones yourself, you will most likely place them in your `~/Library/PreferencePanes` folder. [Figure 8.1](#) shows the System Preferences application's main window with a lot of available preference panes.

Figure 8.1. The System Preferences application.



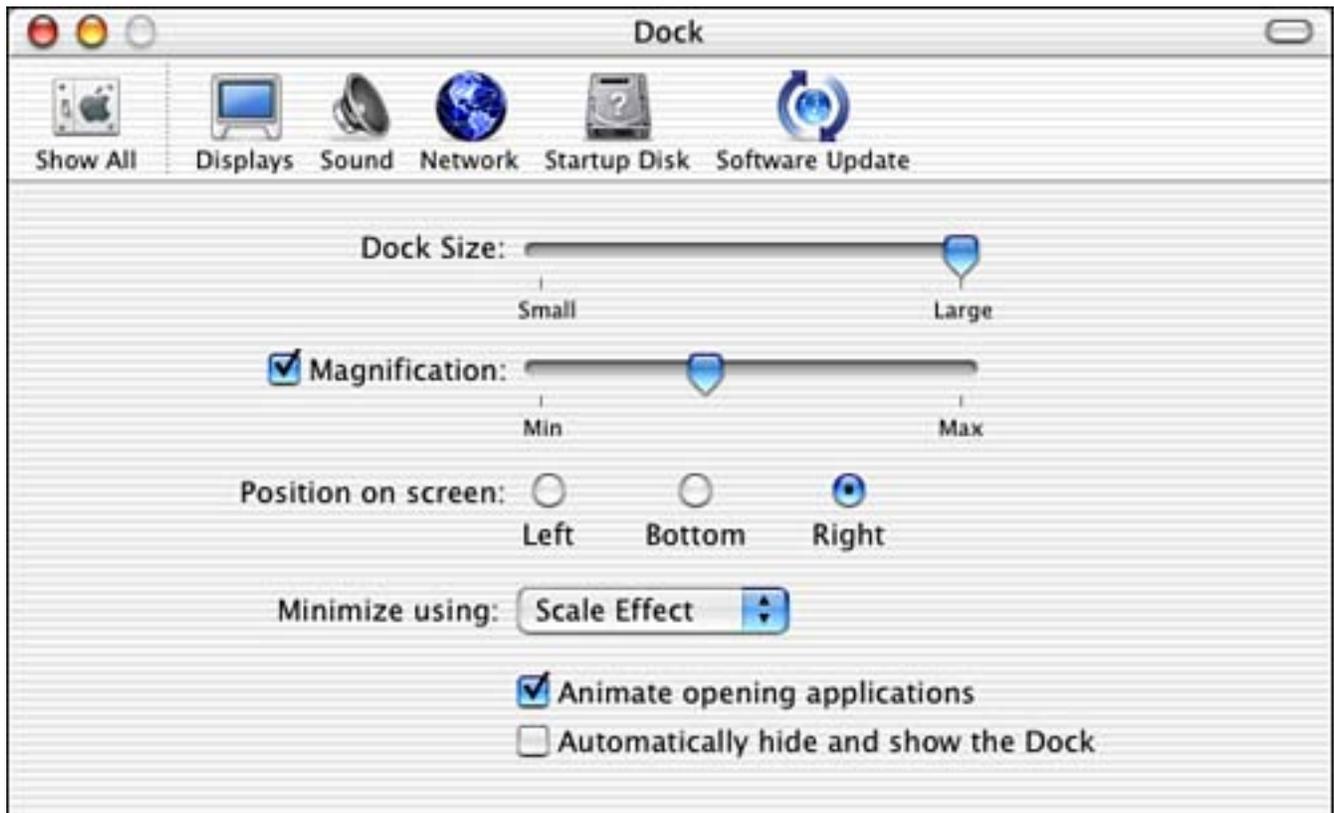
Preference panes are primarily used to control settings for applications that do not have a user interface. This could include a background-only application or some other system-level service that is run at startup. Software Update is a good example: The preference pane allows the user to schedule when updates may occur, and a separate application downloads and installs the latest software. The System Preferences application provides a logical "base of operations" for many types of system functionality, but you should avoid doing real work in a System Preference Pane—that is usually better suited for an application.

Here is another example. [Figure 8.2](#) shows the Dock, and [Figure 8.3](#) shows the Dock preferences as displayed in the System Preferences application. The Dock is the application that runs in conjunction with the Finder, and it displays icons of frequently used and currently running applications. The Dock can function in a variety of ways and allows the user to edit its settings via its preference pane. The Dock also has a power-user shortcut to get to its preferences as well, by control clicking on an empty area in the Dock.

Figure 8.2. Dock.



Figure 8.3. Dock preferences in the System Preferences application.



Note that the Dock preferences all take effect immediately. As soon as the user changes a value in the Dock preferences pane, the change is immediately communicated to the Dock. See [CFNotificationCenter](#) and the Notification Center documentation from Apple for more details about using this technique. Because the Dock is considered a system-level feature that is always available, System Preferences is the most logical place to store and edit its settings.

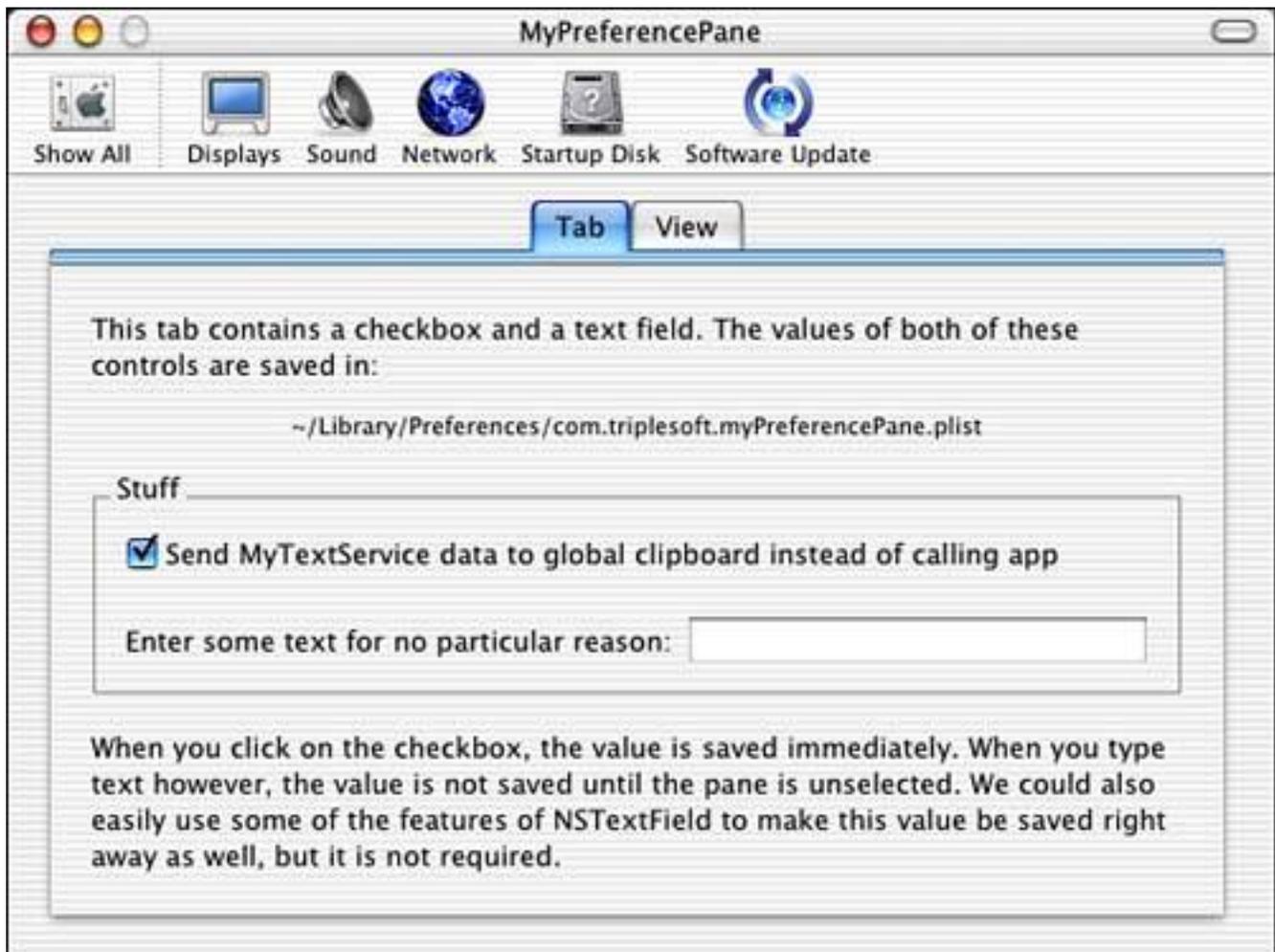
[\[Team LiB \]](#)

◀ PREVIOUS NEXT ▶

The Result

In this chapter, I will discuss a preference pane called `MyPreferencePane`. As I alluded to in the previous chapter, our preference pane will control one of the settings (the only one actually) of the `MyTextService` service. By default, `MyTextService` will paste the altered text directly into the document that has the current selection when it is executed. Our preference allows the altered text to be placed on the global pasteboard instead so that it can be pasted elsewhere, leaving the selected text untouched. [Figure 8.4](#) shows the first tab of `MyPreferencePane`'s preference pane.

Figure 8.4. The `MyPreferencePane` tab in the System Preferences application.



Note that although there are a few different types of controls on this tab, only the check box is used to control the `MyTextService` service preference. The `NSTextField` is merely there to show that other types of data can be managed as a preference as well. As this project is examined, you will see that the preferences are stored in a file located at `~/Library/Preferences/com.triplesoft.myPreferencePane.plist`, which is updated immediately when the check box state changes. That is, there is no need to "save changes" or close the preference pane before the changes take effect. This property list file is shown in [Listing 8.1](#). You can see the `Boolean Value Key` for the check box and the empty `String Value Key` for the `NSTextField`. When `MyTextService` is called, it looks up the current value of the `Boolean Value Key` to decide which pasteboard to use (refer to [Listing 7.3](#) in [Chapter 7](#), "System Services").

Listing 8.1 com.triplesoft.myPreferencePane.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>Boolean Value Key</key>
    <true/>
    <key>String Value Key</key>
    <string></string>
  </dict>
</plist>
```

Note

The Mac OS X terminal contains a command named `defaults` that advanced users and developers should take note of. This command allows you to quickly and easily access the Mac OS X user defaults system to read and write user defaults. Open the Terminal and type `man defaults` for more information.

[Figure 8.5](#) shows the second tab of the MyPreferencePane preference pane. This tab serves no purpose other than to show how simple it is to create a tabbed user interface in Interface Builder and link the items in each tab to actions and outlets in a class. The Click To Beep button, although on a completely different tab in the user interface, is treated no differently when connected in Interface Builder. The Cocoa framework transparently handles all the details of switching tabs for you. You can have as many tabs as you like with as many items in each tab—this is a huge improvement over what developers used to have to do for a multi-tabbed interface.

Figure 8.5. The MyPreferencePane View tab in the System Preferences application.



Note in [Figure 8.1](#) that MyPreferencePane appears at the bottom of the System Preference application window in the Other category. This is where any third-party preference panes appear. You simply click it to make it active.

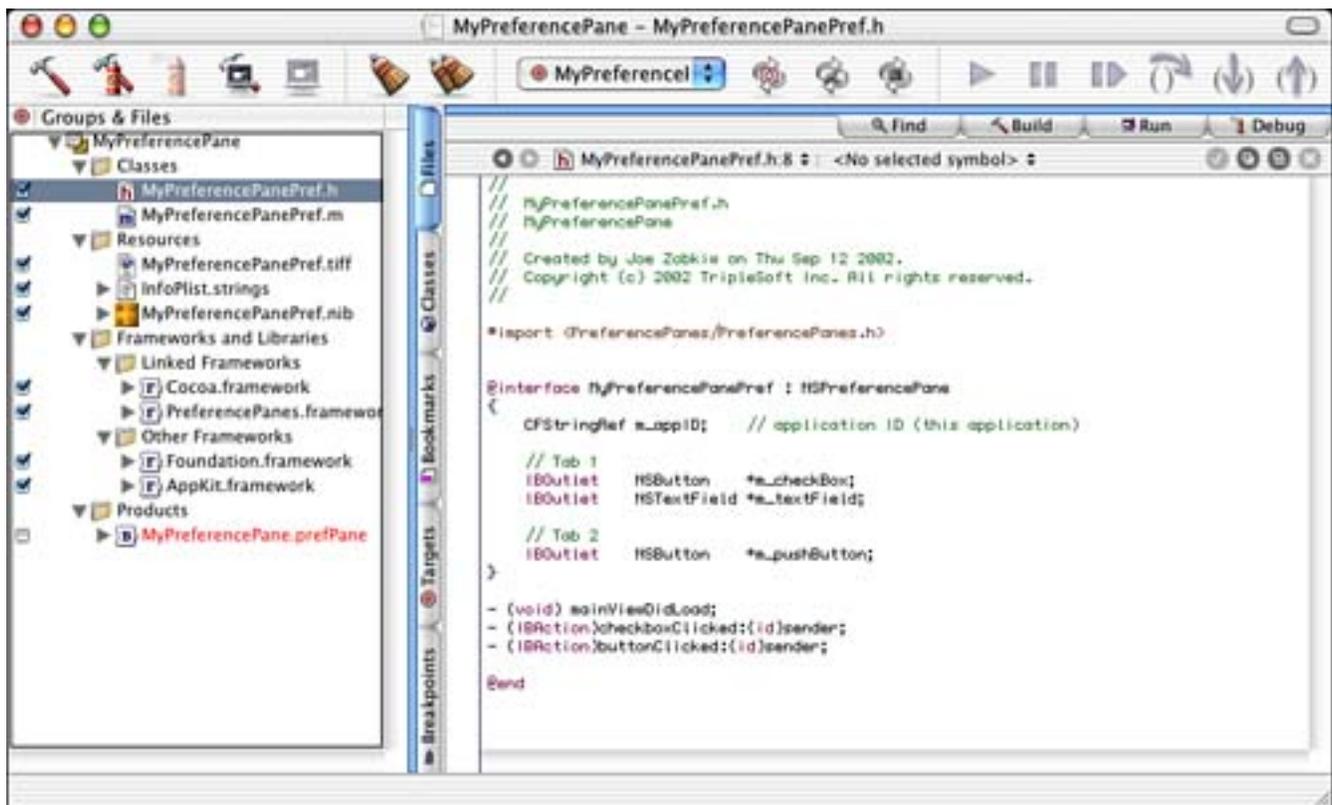
Let's examine the project file for MyPreferencePane!

[\[Team LiB \]](#)

The Project

Figure 8.6 shows the MyPreferencePane project in Project Builder. You can create this project using the Preference Pane template in the New Project Assistant. At a glance, you will notice the `MyPreferencePanePref.h` and `MyPreferencePanePref.m` files that contain the `NSPreferencePane` subclass that is our preference pane. The project also includes a `.tiff` file that contains the icon used to represent our preference pane in the System Preferences application (see Figure 8.1). The `InfoPlist.strings` and `MyPreferencePanePref.nib` files are also present.

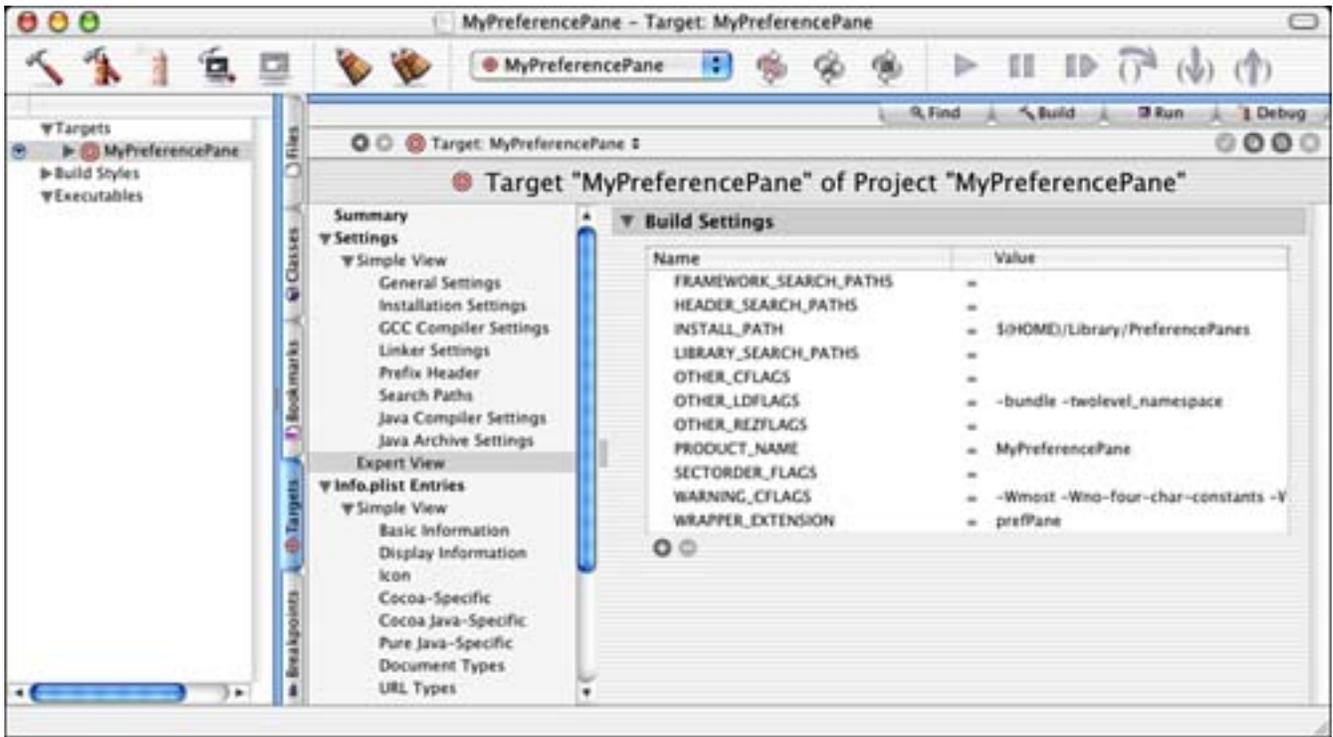
Figure 8.6. The MyPreferencePane project.



The Project Settings

There is probably nothing for you to change in the Target settings, depending on whether things have changed in the project template as of this writing. One thing to note, however, is that the `WRAPPER_EXTENSION` for a preference pane is `prefPane`. Once again, this is how the operating system will tell that this bundle is a preference pane when searching through the `PreferencePanes` folder (see Figure 8.7).

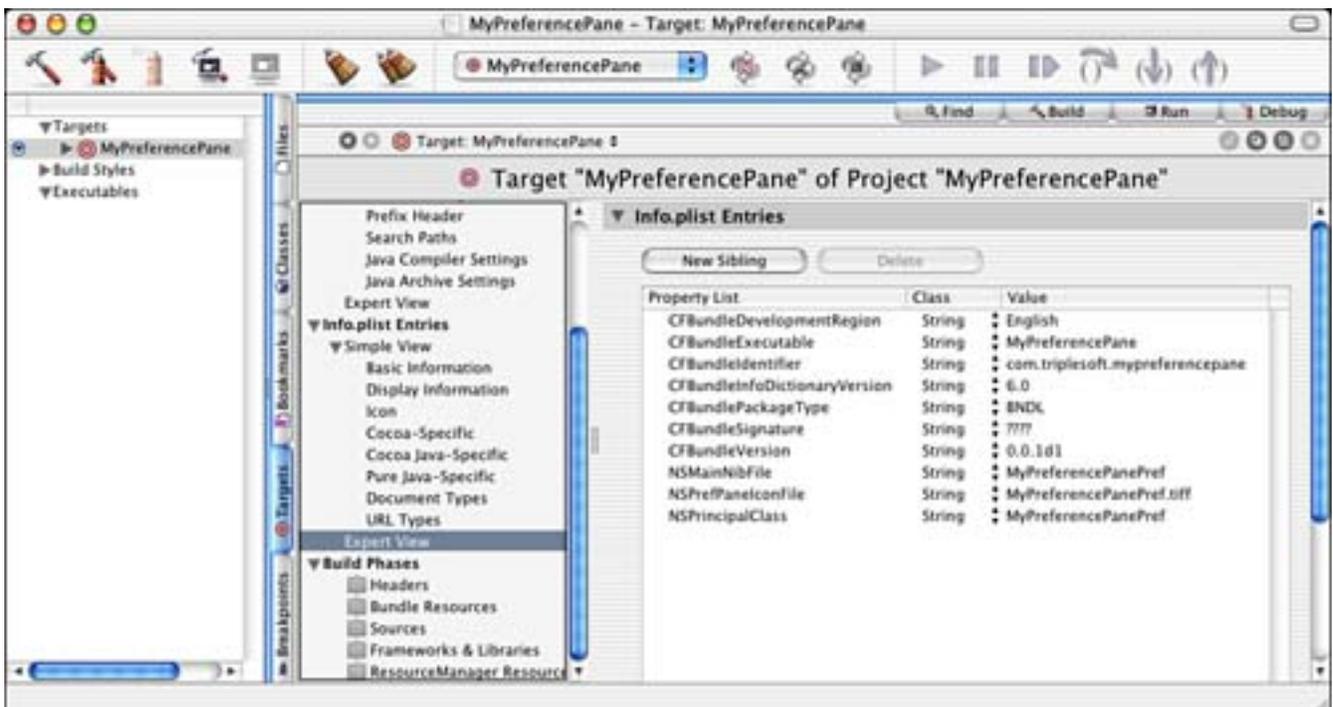
Figure 8.7. The MyPreferencePane Target settings.



Let's look at the project settings.

The InfoPlist settings are relatively straightforward as well in [Figure 8.8](#). You should note the settings of the `CFBundleExecutable` being our filename, `NSMainNibFile` being the name of our nib file, `NSPrefPanelIconFile` being the name of the icon file, and `NSPrincipalClass` being our classname.

Figure 8.8. The MyPreferencePane InfoPlist entries.

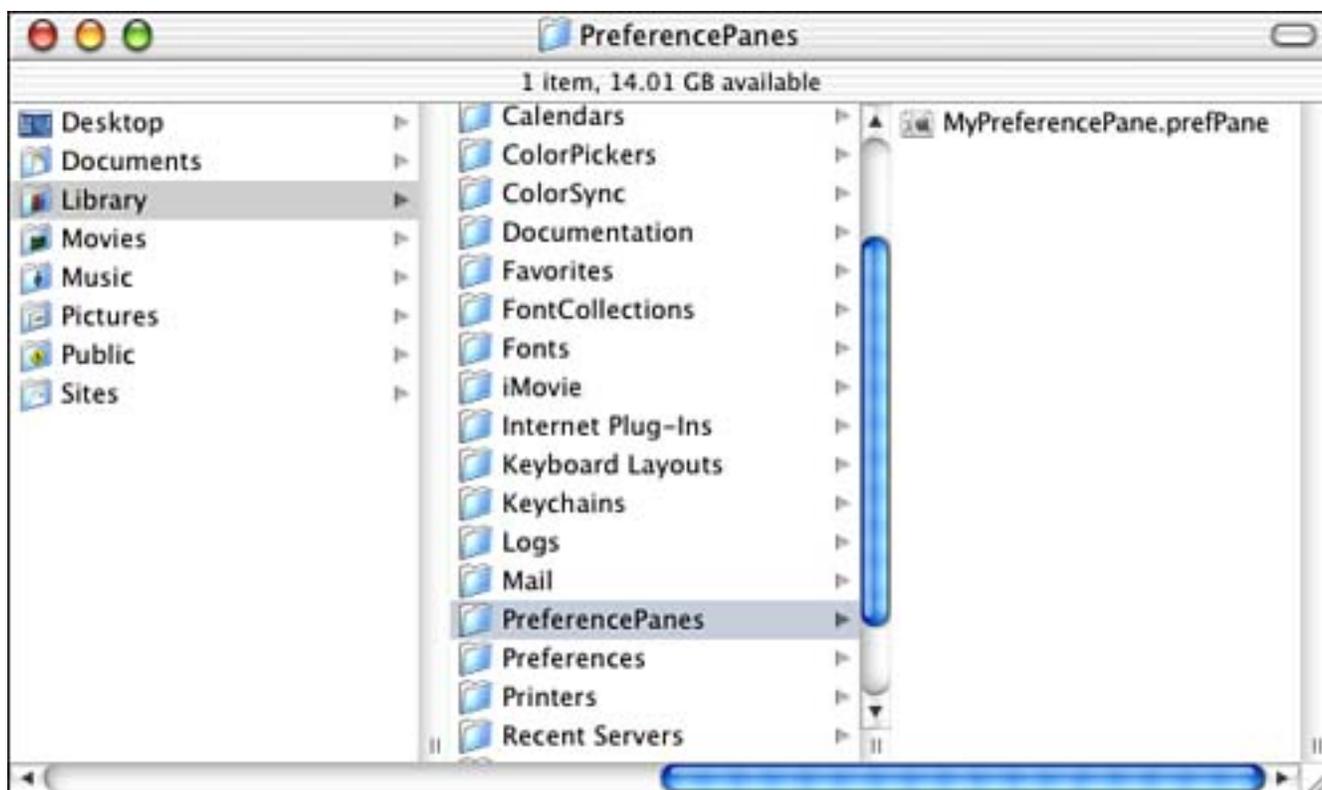


Note

When dealing with TIFF files, you should make sure that their names end in `.tiff` and not `.tif`. Although many programs treat both extensions as a TIFF file, Project Builder and the operating system do not. Mac OS X likes the extra `f`.

Note also that we will place our built preference pane in the `~/Library/PreferencePanes` folder. You might have to create this directory if it does not exist. Also, notice the custom icon that our preference pane has in the Finder. This icon does not come free. The easiest way to apply this icon is to use the `MyPreferencePane.icns` file, included with the source code, to copy and paste the icon into place. By selecting Get Info on the `.icns` file and copying the icon, you can then easily select Get Info on the `.prefPane` file and paste it. The key to remember here is that the `.tiff` file in our project is for use inside the System Preferences application, not in the Finder (see [Figure 8.9](#)).

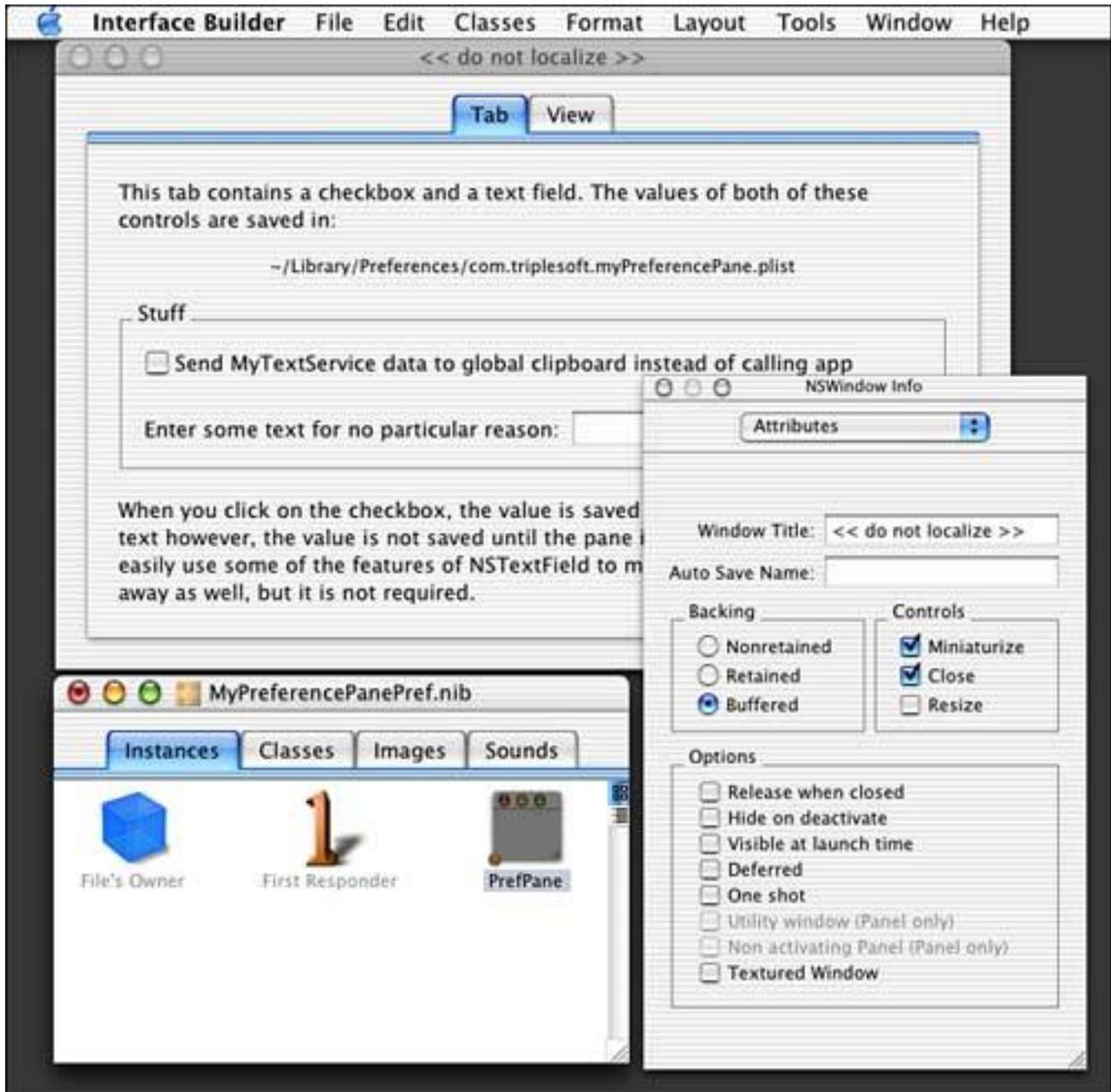
Figure 8.9. The PreferencePanes directory in our `~/Library` directory.



The Nib File

Before we look at the source code, let's peek into the `MyPreferencePanePref.nib` file (see [Figure 8.10](#)). This standard nib file contains an `NSWindow` named `prefPane` that contains the entire user interface of our preference pane. The File's Owner is the `MyPreferencePanePref` class, which contains all the action and outlet instance variables. By assigning the `prefPane` window as the `_window` outlet of the File's Owner, we set up the relationship needed by the System Preferences application to find the views to display when our preference pane is selected. Open the file in Interface Builder to peruse the relationships of the objects for yourself.

Figure 8.10. The `MyPreferencePane` `MyPreferencePanePref.nib` file.



Note

The preference pane window in System Preferences has specific size requirements. The width is fixed to 595 pixels; the height should not cause the window to extend beyond the bottom of an 800x600 display. To be safe, you should not create a preference pane that is more than approximately 500 pixels in height. Make sure that you check your work on a small screen before you ship!

The Source Code

The `MyPreferencePanePref` class is a subclass of `NSPreferencePane`. `NSPreferencePane` is defined in

`PreferencePan.es.h` and implements the default behavior of a preference pane. By overriding methods in `NSPreferencePane`, you customize the behavior of your own preference pane (see [Listing 8.1](#)).

Listing 8.1a `MyPreferencePanePref` Interface in `MyPreferencePanePref.h`

```
#import <PreferencePan.es/PreferencePan.es.h>

@interface MyPreferencePanePref : NSPreferencePane
{
    CFStringRef m_appID;    // application ID (this application)

    // Tab 1
    IBOutlet NSButton *m_checkBox;
    IBOutlet NSTextField *m_textField;

    // Tab 2
    IBOutlet NSButton *m_pushButton;
}

- (IBAction) checkBoxClicked: (id) sender;
- (IBAction) buttonClicked: (id) sender;

@end
```

You will note that our subclass simply keeps track of our application ID (`com.triplesoft.mypreferencepane` from our InfoPlist entries) and our outlets as far as instance variables go. We also implement a few actions to handle our check box and button. Note that even though we have a multi-tabbed user interface, our outlets and actions are all defined as usual—no special treatment is required.

Note

If you choose to not cache the bundle identifier, you can use `[[NSBundle bundleForClass:[self class]] bundleIdentifier]` to look it up on demand. If you end up accessing this information regularly throughout your preference pane, you might consider the cache method shown here.

[Listing 8.2](#) is an override of the `-initWithBundle:` method of `NSPreferencePane`. This method is called during the initialization phase of the preference pane. Its primary goal is to keep track of our application ID in the `m_appID` instance variable. You can also perform any other initialization here that does not depend on the user interface.

Listing 8.2 `MyPreferencePanePref -initWithBundle:` Implementation in `MyPreferencePanePref.m`

```
// This function is called as we are being initialized
- (id) initWithBundle: (NSBundle *) bundle
{
    // Initialize the location of our preferences
    if ((self = [super initWithBundle: bundle]) != nil) {
```

```

        m_appID = CFSTR("com.triplesoft.myreferencepane");
    }

    return self;
}

```

[Listing 8.3](#) contains the override of the `-mainViewDidLoad` method of `NSPreferencePane`. This method is called once the `Nib` is loaded and the windows have been created. You would perform tasks in this method similar to those that you would in `-awakeFromNib` in an application. We have two preferences to load and set at this point. Using the `CFPreferencesCopyAppValue` function, we load both `Boolean Value Key` and `String Value Key` and set the `NSButton` check box and `NSTextField` appropriately. Note that we also verify the data coming back from this function using the `CFGetTypeID` function in conjunction with the `CFBooleanGetTypeID` and `CFStringGetTypeID` functions. This provides a reliable way for us to ensure that our data is not corrupt in any way. Last, the values of the outlets are set accordingly to display the current state of the preferences. Remember to `CFRelease` the values once they are loaded!

Listing 8.3 MyPreferencePanePref -mainViewDidLoad Implementation in MyPreferencePanePref.m

```

// This function is called once the Nib is loaded and our windows
// are ready to be initialized.
- (void)mainViewDidLoad
{
    CFPropertyListRef value;

    // Load the value of the checkbox as a BOOLEAN
    value = CFPreferencesCopyAppValue(CFSTR("Boolean Value Key"), m_appID);
    if (value && CFGetTypeID(value) == CFBooleanGetTypeID()) {
        [_checkBox setState:CFBooleanGetValue(value)];
    } else {
        [_checkBox setState:NO]; // Default value of the checkbox
    }
    if (value) CFRelease(value);

    // Load the value of the text field as a STRING
    value = CFPreferencesCopyAppValue(CFSTR("String Value Key"), m_appID);
    if (value && CFGetTypeID(value) == CFStringGetTypeID()) {
        [_textField setStringValue:(NSString *)value];
    } else {
        [_textField setStringValue:@""]; // Default value of the text field
    }
    if (value) CFRelease(value);
}

```

Note

Remember, `.plist` files are nothing more than plain text in XML format. These files are very easy for a user to open up and "mess around with." It doesn't hurt to err on the side of caution and verify that the values retrieved are the types of values we expect.

[Listing 8.4](#) implements the override of the `-di dUnsel ect` method of `NSPreferencePane`. This function is called when our preference pane is being deselected. That is, either another preference pane is being selected or the System Preferences application is shutting down. In this case, we save the contents of our `NSTextFi el d` outlet, which is not saved at any other time. You could implement a notification to tell each time the user typed a character in the `NSTextFi el d` and save then, but that would be overkill in this situation.

Listing 8.4 `MyPreferencePanePref -di dUnsel ect` Implementation in `MyPreferencePanePref.m`

```
// This function is called when our preference
// pane is being deselected. That is, either
// another preference pane is being selected
// or the System Preferences application is
// shutting down. In this case we will save our
// text field, which is not saved at any other
// time. The checkbox is saved as it is clicked
// but just as easily can be saved here.
- (void)di dUnsel ect
{
    CFNotifi cationCenterRef center;

    // Save text field
    CFPreferencesSetAppVal ue(CFSTR("String Val ue Key"),
        [m_textFi el d stringValue], m_appID);

    // Write out all the changes that have been made for this application
    CFPreferencesAppSynchroni ze(m_appID);

    // Post a notification that the preferences
    // for this application have changed,
    // any observers will then become the first
    // to know that this has occurred.
    center = CFNotifi cationCenterGetDi stri butedCenter();
    CFNotifi cationCenterPostNotifi cation(center,
        CFSTR("Preferences Changed"), m_appID, NULL, TRUE);
}
```

Note that the string value of the `NSTextFi el d` is saved using the `CFPreferencesSetAppVal ue` function. Once saved, the `CFPreferencesAppSynchroni ze` function is called to flush the contents of the file and be sure that they are written to disk. You will note from [Listing 7.2](#) in [Chapter 7](#) that `MyTextService` also calls `CFPreferenceAppSynchroni ze` before accessing the preferences. This magic keeps everyone happy and "in sync."

The last step is optional and is shown here as an example. We post a notification using `CFNotifi cationCenterPostNotifi cation` and `CFNotifi cationCenterGetDi stri butedCenter` to let any observers know that the preferences have changed. Even though `MyTextService` does not implement this mechanism, it is not a bad idea to get into the habit of using it should some other application in your suite be "watching" your preferences. An application normally won't do this, but because we are working on a globally available systemwide feature, we do. Earlier in this chapter, I discussed where to find more information on using notifications.

[Listing 8.5](#) contains the action that is called when the user clicks on the `NSBut ton` check box. Each time the user clicks the check box; the value is changed and is written to the preference file. Note that the

preferences are once again synchronized after being set.

Listing 8.5 `MyPreferencePanePref -checkboxClicked:` Implementation in `MyPreferencePanePref.m`

```
// This action is called when our checkbox is clicked,
// we save the value immediately
- (IBAction)checkboxClicked: (id) sender
{
    CFPreferencesSetAppValue( CFSTR("Boolean Value Key"),
        [sender state] ? kCFBooleanTrue : kCFBooleanFalse, m_appID);

    // Force a synchronization so our preference is "live"
    CFPreferencesAppSynchronize(m_appID);
}
```

Note

What? You've never seen that `? :` thing before? I love this feature of the C programming language and most C programming language derivatives. This mechanism is called a conditional expression. The format is as follows:

```
A ? B : C;
```

The first expression, `A`, is evaluated. If `A` is `TRUE`, `B` is executed; otherwise `C` is executed. This is equivalent to the following:

```
if (A) {
    B;
} else {
    C;
}
```

In our case, we are using the conditional operator to return one of two different values: if the `-state` of the `sender` (the check box) is `YES` (also known as `1`), return `kCFBooleanTrue`; otherwise return `kCFBooleanFalse`.

Finally, in [Listing 8.6](#) we have an action that is called when the button on the second tab is clicked (refer to [Figure 8.5](#)). This action is here to show you that the same class can be used to handle actions and outlets on multiple tabs in a multi-tabbed user interface. Clicking the Click To Beep button makes the computer beep.

Listing 8.6 `MyPreferencePanePref -buttonClicked:` Implementation in `MyPreferencePanePref.m`

```
// This action is called when the button on the second tab is clicked
- (IBAction)buttonClicked: (id) sender
{
```

```
    NSBeep();  
}
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

More NSPreferencePane

Many other methods are defined in `NSPreferencePane` that can be overridden by your subclass. We only implement a few of them in this project because that is all that was necessary. Some of the others that you may consider overriding include

- `assignMainView`— This method can be overridden if your `NSPreferencePane` subclass needs to dynamically choose a main view. You might need to do this to show a unique set of preferences based on the current hardware configuration.

- `willSelect`/`didSelect`— These methods can be overridden if you need to be called just before or just after your `NSPreferencePane` subclass becomes the currently selected preference pane. You might need to do this to load a data file or begin some task.

- `willUnselect`/`didUnselect`— These methods can be overridden if you need to be called just before or just after your `NSPreferencePane` subclass get swapped out as the currently selected preference pane. You might need to do this to save any preferences that are not saved as they are changed.

- `shouldUnselect`/`replyToShouldUnselect`: — These methods allow your `NSPreferencePane` subclass to postpone or cancel an unselect action. For example, if your preference pane begins a network action and then the user attempts to choose another pane, you might want to delay the closing of your pane until the network action completes or can be cancelled.

- `initialKeyView`/`firstKeyView`/`lastKeyView`— This method can be overridden if your `NSPreferencePane` subclass needs to dynamically determine which view should be the initial, first, or last key view, respectively.

Be sure to look at `NSPreferences.h` for more details on other methods and some inside information on how they operate.

Try This

Here are some ideas to expand the preference pane project in this chapter; try these on your own.

Add a third tab to the preference pane and some new controls such as radio buttons and a pop-up button. Save the settings of these controls to the preference file as appropriate. Make sure to test them to see that they are really saving!

Add an `NSTextField` control below the tabs so that it is always displayed whenever the preference pane is selected. Make use of the `NSTimer` class you learned about in [Chapter 3](#), "Cocoa Applications," and have this control display how long, in seconds, that the preference pane has been visible. Be sure to properly dispose of any timer that has yet to "fire" when the user closes the preference pane.

Conclusion

Now you see that we can add preferences to a service by creating a preference pane project to go along with our system service project. Once you know the secrets, these are just as straightforward as implementing any Cocoa application. All similar pieces come together with just a few differences.

In this case, I asked you to drag the `.prefPane` file into the `PreferencePanes` folder. Normally, you would create an installer to install these items without intimate knowledge by the user. You don't want users dragging `.service` and `.prefPane` files around; you never know when they will drop them in the wrong location and then complain that your software isn't working.

Keep up the good work—you've learned a lot so far!

Chapter 9. Status Items

"Should? Will. Should? Will. Should? Will. Did!"

—quote based on Cocoa delegate methods

This chapter discusses those little black and white icons on the right side of the menu bar as displayed in [Figure 9.1](#). Before we assign these items a real name, let's make something perfectly clear: They are not all created equal. Those created by Apple that can be command dragged to swap locations and added simply by dropping a bundle on the menu bar are called `NSMenuExtras`. However, the `NSMenuExtra` class is a private API, so, as developers who do not work for Apple, we need to create `NSStatusItems` instead.

Figure 9.1. The menu bar filled with "little black and white icons."



Also, Apple says status items are reserved for Apple's use only (see *Aqua Human Interface Guidelines*, page 63). Further, these items are intended to display hardware or network status/settings only. This is why you will see iChat and Airport but not Mail.app. Having said that, third-party developers should respect this guidance by using status items to communicate hardware or network information only. For example, a disk light application seems acceptable, but a Web browser may not be. The example shown in this chapter is an example of the technology, not what to do with it.

`NSStatusItems` aren't so bad in and of themselves. True, they are the not-so-coordinated cousins of the `NSMenuExtra`, but they can get the job done. As mentioned, you can't move them around the menu bar by command dragging. You also cannot add them to the menu bar by dropping their bundle on it. They also don't work perfectly if you click down elsewhere in the menu bar and then drag over the `NSStatusItem`—you have to click directly on the `NSStatusItem` to have them show their menu contents. Other than these few idiosyncrasies, if you need an item to display status in the menu bar, this is the way to do it. For sake of argument, I will refer to items in the menu bar as *status items* even though you know that this is not completely correct.

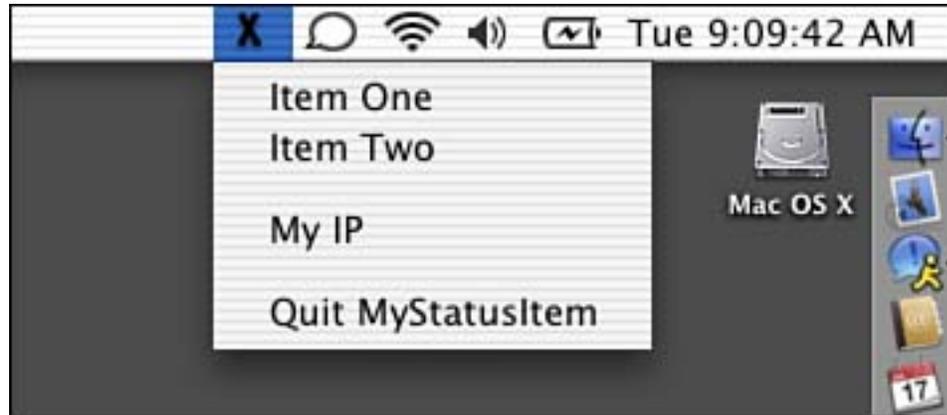
Note

If you're in a hacking mood, there are ways of making `NSMenuExtra` talk. That is, some programmers have reverse-engineered the API and used it in non-Apple products. This is risky business because you never know when Apple will alter the API just enough to cause your software not to work anymore. Feel free to experiment, but keep this in mind before releasing any software that uses undocumented features of the OS.

And remember...

1. Apple claims status items are for its use only.
2. Apple says status items are for reporting hardware or network settings only. See [Figure 9.2](#) for some examples.

Figure 9.2. The menu of the MyStatusItem status item.



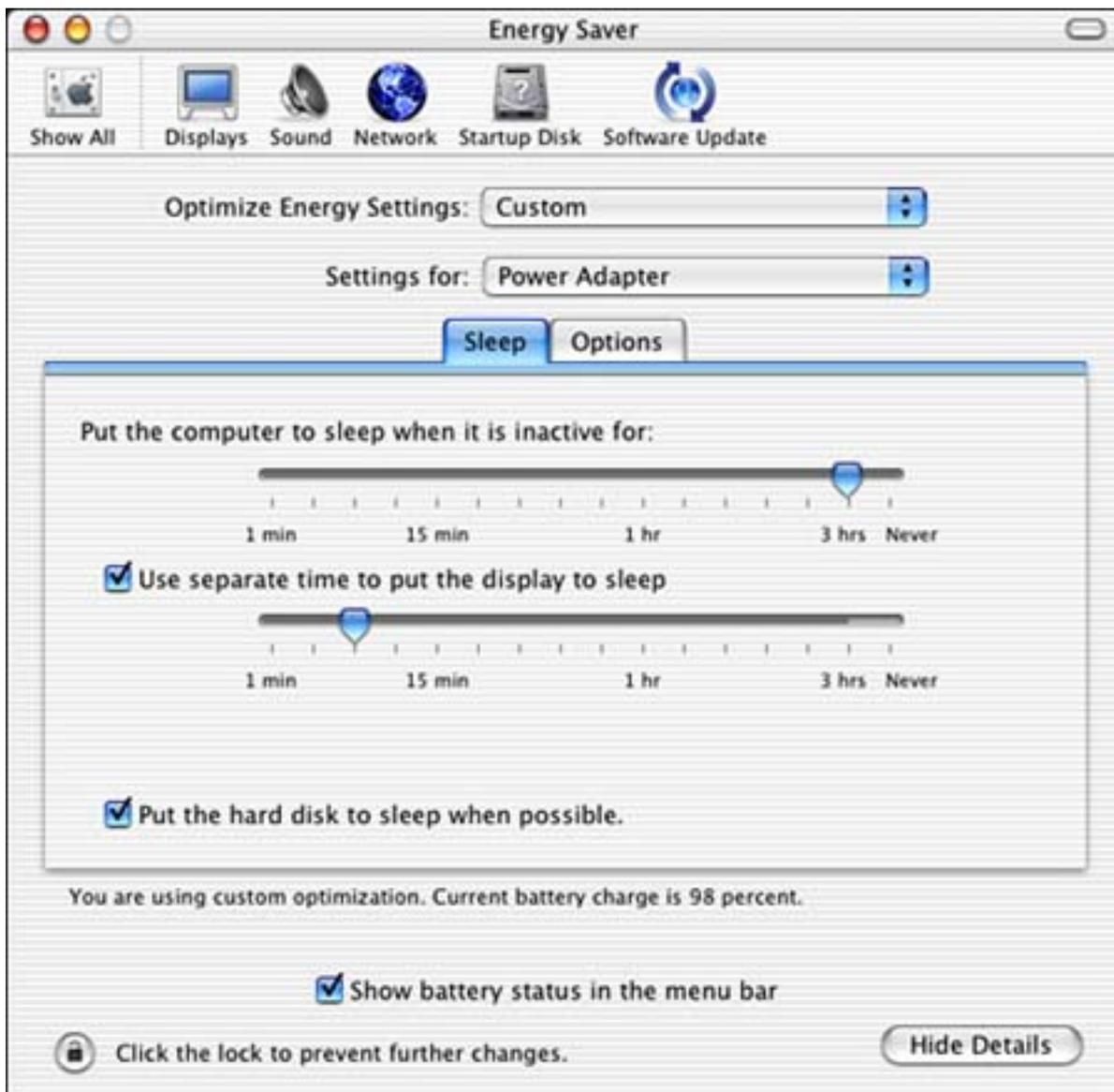
3. Applications should not create/use status items for displaying globally available menu commands. Instead, applications should use Dock menus. See [Chapter 15](#), "SOAP," for more information on creating and managing Dock menus.

The Result

[Figure 9.2](#) shows a menu bar with a variety of status items. The one on the left is the MyStatusItem status item that we will be examining in this chapter. Note that when you click on it, its menu is displayed. The menu contains four items—six if you count the item separators. The first two cause an alert to be shown, the third displays the local IP address of the computer, and the fourth causes the program to quit, making the menu and status item disappear from the menu bar. We will look at these in more detail shortly.

Other status items in the Jaguar menu bar, from left to right, include those for iChat, AirPort, Volume, Battery, and the Date and Time. You might notice that many of these are controlled via an option in a Preference Pane as to whether or not they appear. [Figure 9.3](#) shows the Energy Saver Preference Pane. Note that at the bottom of the window there is a check box entitled "Show Battery Status in the Menu Bar." If this check box is turned on, the battery status item will display in the menu bar. If it is turned off, the battery status item disappears.

Figure 9.3. The Energy Saver preference pane offers the option to display the battery status in the menu bar.



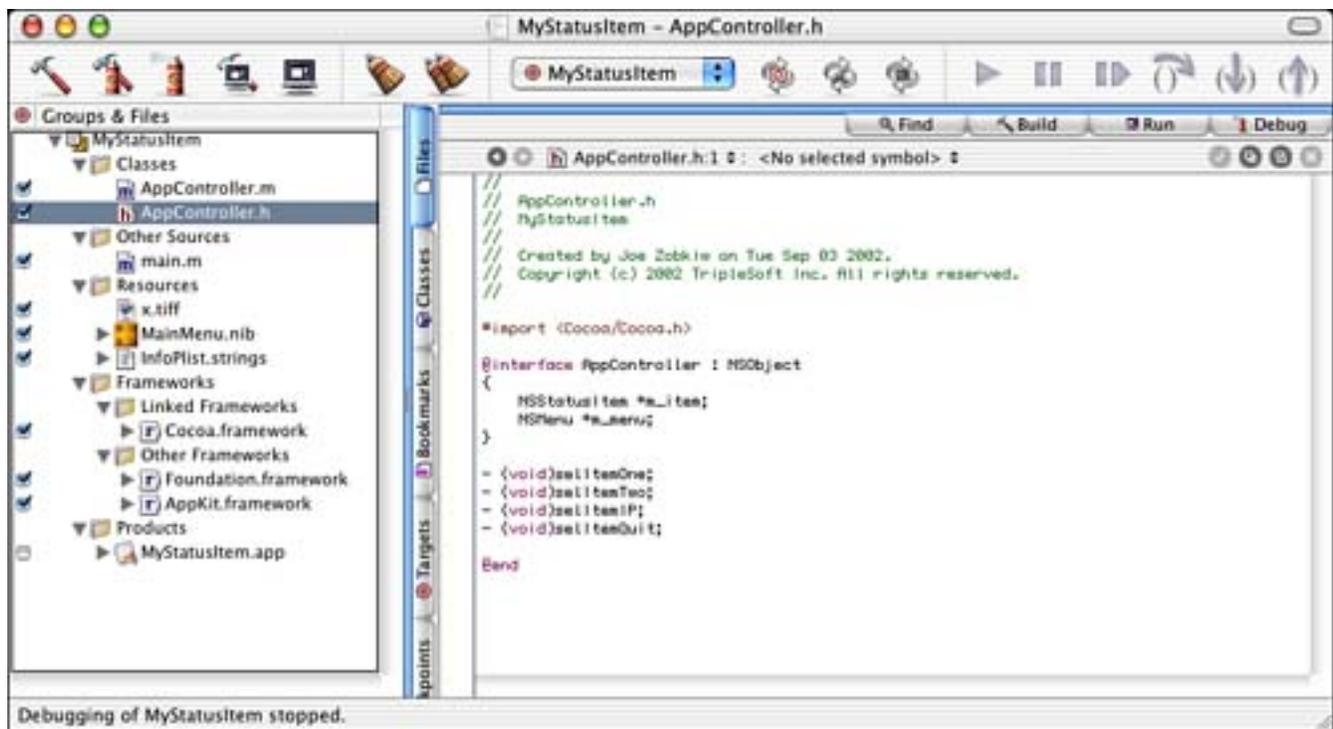
Let's look at the project that makes MyStatusItem appear in the menu bar.

[\[Team LiB \]](#)

The Project

[Figure 9.4](#) shows the MyStatusItem project in Project Builder. This project was created by starting with the standard Cocoa application template in the New Project Assistant. You will notice that the only source code files other than the standard items are `AppController.m` and `AppController.h`. All of our work is done in our `AppController` class although you could easily place it somewhere else depending on the needs of your application. We also have a `TIFF` file that we added to the project that contains the X image used in the menu bar. `MainMenu.nib` has hardly been touched with the exception that we filled out our window with some text that will be displayed when you run the application and linked our `AppController` to the File's Owner (`NSApplication`) as its delegate. Any other changes to `MainMenu.nib` are negligible and not pertinent to the project.

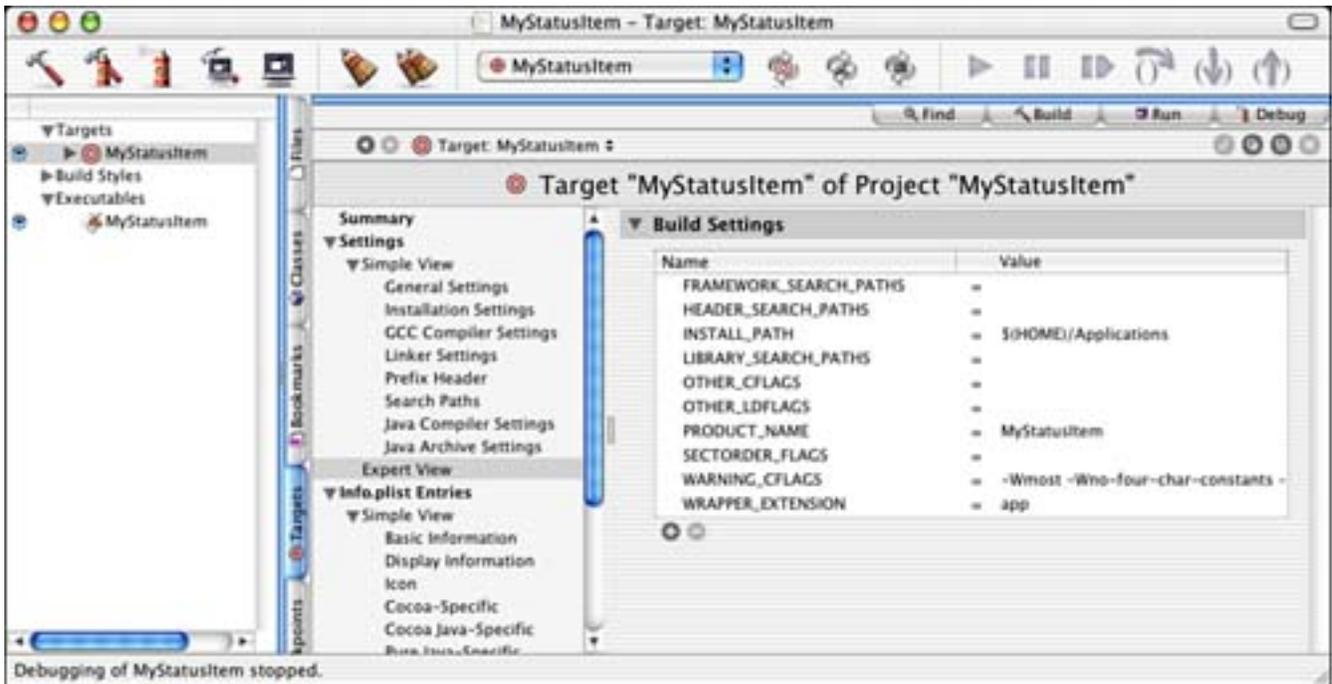
Figure 9.4. The MyStatusItem project.



The Project Settings

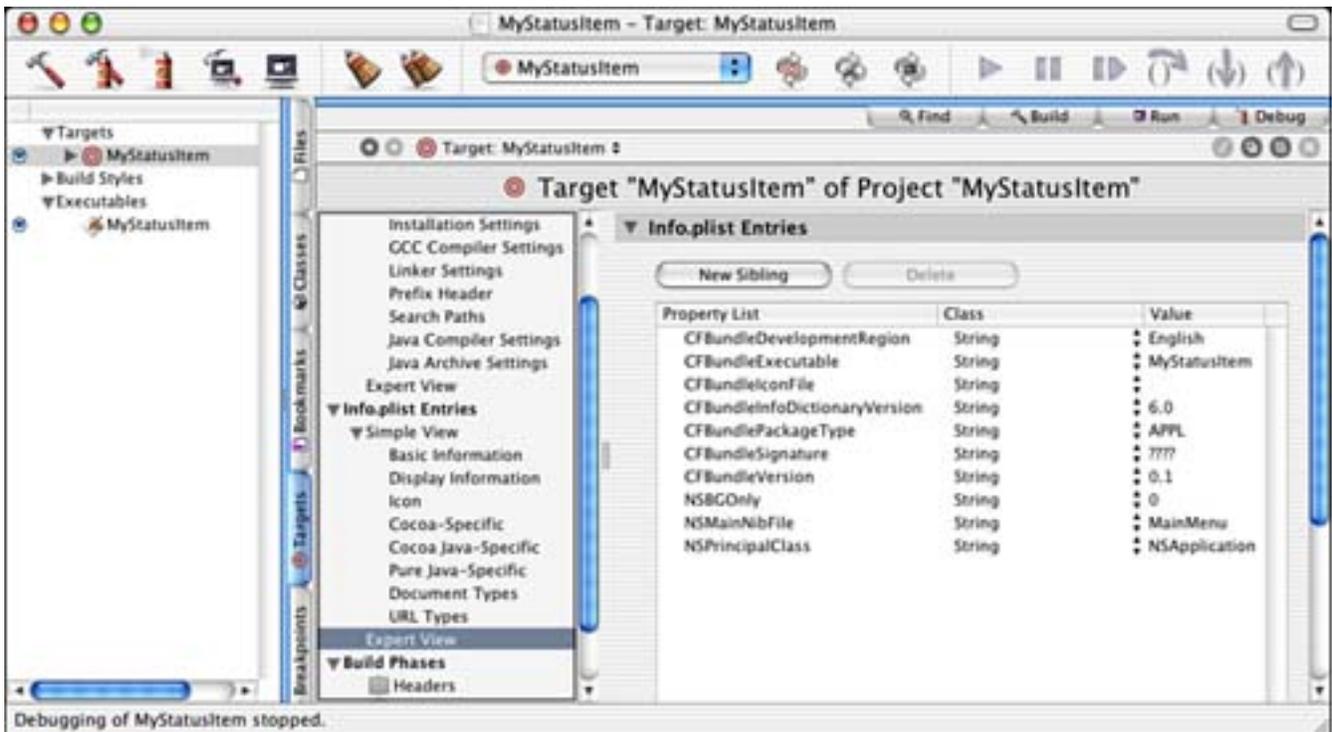
There isn't much to say about the Target Settings in [Figure 9.5](#)—status quo here.

Figure 9.5. The MyStatusItem Target settings.



The InfoPlist Entries in [Figure 9.6](#) are also about as straightforward as you can get—standard for a Cocoa application. The one possible exception is that we've added the `NSBGOnly` entry. This entry is currently set to `0`, so it is essentially disabled. You might remember from earlier chapters that setting this entry to `1` causes our application to become a background-only application; that is, with no user interface, per se. We will look into this in more detail later in the chapter.

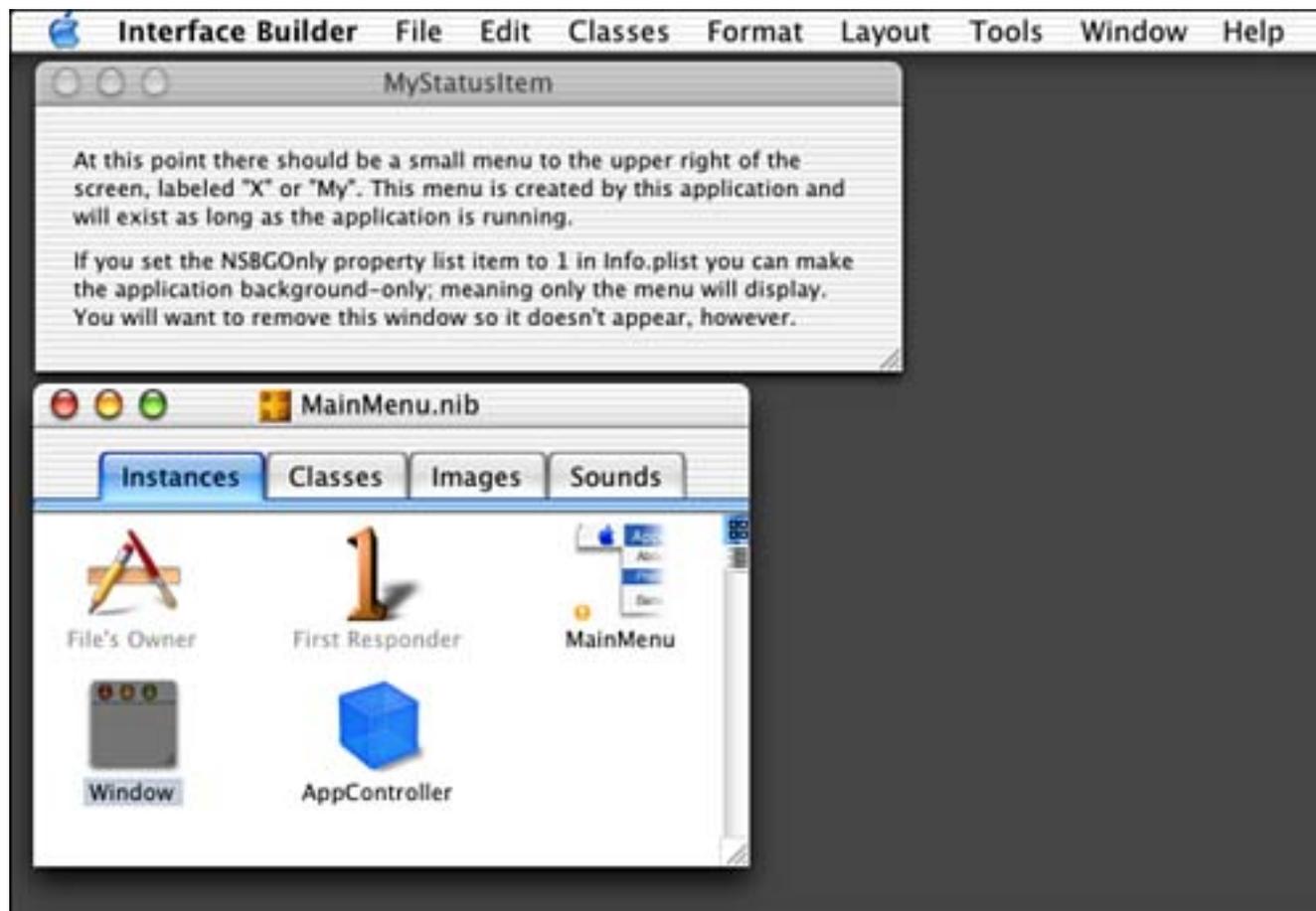
Figure 9.6. The MyStatusItem InfoPlist entries.



The Nib File

As mentioned previously, our `MainMenu.nib` file is also straightforward. We created a window that appears when the application is launched. [Figure 9.7](#) displays the window as shown in Interface Builder. If you want to experience the literary art of foreshadowing, read the text of the window in [Figure 9.7](#).

Figure 9.7. The MyStatusItem MainMenu.nib file.



Let's jump into the source code.

The Source Code

`AppController.m` is our only custom source file in this project. We have the `AppController` subclass of `NSObject`, which is defined as shown in [Listing 9.1](#).

Listing 9.1 `AppController` Interface in `AppController.h`

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

@interface AppController : NSObject
{
    NSStatusItem *m_item;
    NSMenu *m_menu;
}

- (void) selItemOne: (id) sender;
```

```
- (void) selItemTwo: (id) sender;
- (void) selItemIP: (id) sender;
- (void) selItemQuit: (id) sender;
```

```
@end
```

The only instance variables we need to manage are a pointer to the `NSStatusItem`, the actual item that sits in the menu bar, and the `NSMenu` that is displayed when you click on the `NSStatusItem`. Both of these instance variables are created at runtime in the `AppController`'s `-init` method.

The `-init` method of `AppController` (see [Listing 9.2](#)) is where all things are created. Upon initializing ourselves by calling our super class, we begin the process of creating the `NSStatusItem` and `NSMenu`. First, we call the `+systemStatusBar` class method of `NSStatusBar`. This will return the global system `NSStatusBar` to which we will attach our `NSStatusItem`. The system status bar is the only status bar that is currently available to add items to, as of this writing.

Listing 9.2 `AppController -init` Method in `AppController.m`

```
- (id) init
{
    [super init];
    if (self) {
        // Get the system status bar
        NSStatusBar *bar = [NSStatusBar systemStatusBar];

        // Add an item to the status bar
#ifdef USE_TEXT_TITLE
        m_item = [[bar statusItemWithLength: NSVariableStatusItemLength]
                 retain];
        [m_item setTitle: NSLocalizedString(@"My", @"")];
#else
        m_item = [[bar statusItemWithLength: NSSquareStatusItemLength]
                 retain];
        [m_item setImage: [NSImage imageNamed:@"x"]];
#endif

        [m_item setHighlightMode: YES];

        // Create a menu to be added to the item
        m_menu = [[[NSMenu alloc] init] retain];
        [m_menu setAutoenablesItems: YES];
        [m_menu addItem: [[NSMenuItem alloc] initWithTitle:
                          NSLocalizedString(@"Item One", @"")
                          action:@selector(selItemOne:) keyEquivalent:@""]];
        [m_menu addItem: [[NSMenuItem alloc] initWithTitle:
                          NSLocalizedString(@"Item Two", @"")
                          action:@selector(selItemTwo:) keyEquivalent:@""]];
        [m_menu addItem: [NSMenuItem separatorItem]];
        [m_menu addItem: [[NSMenuItem alloc] initWithTitle:
                          NSLocalizedString(@"My IP", @"")
                          action:@selector(selItemIP:) keyEquivalent:@""]];
        [m_menu addItem: [NSMenuItem separatorItem]];
        [m_menu addItem: [[NSMenuItem alloc] initWithTitle:
                          NSLocalizedString(@"Quit MyStatusItem", @"")
```

```
action:@selector(selItemQuit:) keyEquivalent:@""]];
```

```
// And add the menu to the item
```

```
[menuItem setMenu:m_menu];
```

```
}
```

```
return self;
```

```
}
```

We then have implemented two mechanisms to create the `NSStatusItem`—one using text as the item title and one using an image. In [Figure 9.2](#), you could see that the XTIFF image represents the status item. However, you can just as easily use text as the item title, as shown in [Figure 9.8](#).

Figure 9.8. Using text as the title of an item.



Depending on the mechanism you choose, you will either pass the `NSVariableStatusItemLength` or `NSSquareStatusItemLength` constant to `NSStatusBar`'s `-statusBarItemLength:` method. Both of these are defined in `NSStatusBar.h`. If you are using text as your title, you need to specify the variable length because text can be any (variable) length. You should probably consider keeping your status item's title short, however, because status bar space is at a premium. The `-statusBarItemLength:` method will return a new `NSStatusItem` that you should `-retain`. Your new item will be added to the left of any preexisting items in the status bar. You will then call either the `-setTitle:` or `-setImage:` methods of `NSStatusItem` on the newly returned status item to label it properly. Before we begin to build the menu that attaches to the status item, we call the `-setHighlightMode:` method to enable the user feedback highlighting of the status item when it is clicked—some people call this inverting.

Next, we must create the menu that is attached to the status item. First we `+alloc`, `-init`, and `-retain` an `NSMenu` item. We then call `NSMenu`'s `-setAutoEnableItems:` method to enable the use of the `NSMenuValidation` informal protocol. This causes our `AppController`'s `-validateMenuItem:` method to be called to determine whether a particular menu item should be enabled or not at any particular time. You will see this method in action shortly.

The next six successive calls to the `-addItem:` method append items to our `NSMenu` instance variable. The first call adds an item named "Item One" and sets the action of the item to our `AppController`'s method `-selItemOne:`. The key equivalent is left empty as is recommended for status items. Remember, a limited number of keyboard equivalents are available, so system-level software should avoid them when capable. We continue adding the other items, and some separator items as well, until our menu is built. The last step in this function is to attach the newly created menu to the status item by calling `NSStatusItem`'s `-setMenu:` method.

The `AppController`'s `-dealloc` method (see [Listing 9.3](#)) handles the `-release` of `NSStatusItem` and `NSMenu`.

Listing 9.3 `AppController -dealloc` Method in `AppController.m`

```
- (void) dealloc
{
    [m_item release];
    [m_menu release];
    [super dealloc];
}
```

The `AppController`'s `-validateMenuItem:` method (see [Listing 9.4](#)) is executed due to our call to `NSMenu`'s `-setAutoEnableItems:`. This method is called once for each item in the menu any time the menu is displayed. This allows us to dynamically enable or disable our items. Returning `YES` causes the menu item to be enabled; `NO` disables it.

Listing 9.4 `AppController -validateMenuItem:` Method in `AppController.m`

```
- (BOOL) validateMenuItem: (NSMenuItem *) menuItem
{
    // NSString *selectorString;
    // selectorString = NSStringFromSelector([menuItem action]);
    // NSLog(@"validateMenuItem called for %@", selectorString);

    if ([menuItem action] == @selector(selItemOne:))
        return YES;
    if ([menuItem action] == @selector(selItemTwo:))
        return YES;
    if ([menuItem action] == @selector(selItemIP:))
        return YES;
    if ([menuItem action] == @selector(selItemQuit:))
        return YES;

    return NO;
}
```

The two methods in [Listing 9.5](#) are the actions for the first two items in the menu—Item One and Item Two. Remember in the `-init` method, we designated action methods for each menu item. These actions merely display an alert to the user to show feedback that we selected the menu item.

Listing 9.5 `AppController -selItemOne:` and `-selItemTwo:` Methods in `AppController.m`

```
- (void) selItemOne: (id) sender
{
    int result;
    NSBeep();
    result = NSRunAlertPanel(@"MyStatusItem",
        @"Thank you for selecting menu item one.", @"OK", @"", @"");
}
```

```

- (void) selItemTwo: (id) sender
{
    int result;
    NSBeep();
    result = NSRunAlertPanel(@"MyStatusItem",
        @"Thank you for selecting menu item two.", @"OK", @"", @"");
}

```

[Listing 9.6](#) shows the action called when the My IP menu item is selected. This method merely uses the `NSHost +currentHost` method and the `-address` and `-name` methods to report the current IP address and hostname to the user. [Figure 9.9](#) shows what this alert looks like when displayed. Note that these numbers represent your computer on your local network and might be different from the IP address that the world sees you as through a router.

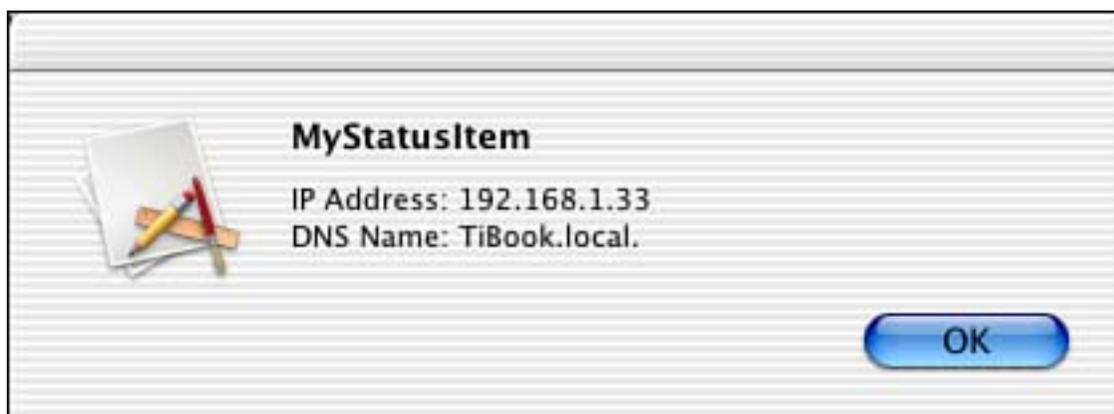
Listing 9.6 `AppController -selItemIP: Method in AppController.m`

```

- (void) selItemIP: (id) sender
{
    int result;
    NSString *theString = [NSString stringWithFormat:
        @"IP Address: %@\nDNS Name: %@",
        [[NSHost currentHost] address], [[NSHost currentHost] name]];
    NSBeep();
    result = NSRunAlertPanel(@"MyStatusItem", theString, @"OK", @"", @"");
}

```

Figure 9.9. The My IP alert.



Finally, when the user selects Quit MyStatusItem from the status item menu, the `-selItemQuit:` method is called as shown in [Listing 9.7](#). After confirming the user's intention to quit, the `NSApplication's +sharedApplication` class method is called to return the instance of your application. Subsequently calling the `-terminate:` method causes the application to quit. If you need to override this mechanism, you can implement the `-applicationShouldTerminate:` method to have one last chance to cancel the termination. If you need to clean up after your application, you can implement the `applicationWillTerminate:` method to perform those tasks.

Listing 9.7 `AppController -selItemQuit: Method in AppController.m`

```
- (void) selItemQuit: (id) sender
{
    int result;
    NSBeep();
    result = NSRunAlertPanel(@"MyStatusItem",
        @"Quit the application and remove this menu?",
        @"Yes", @"No", @"");
    if (result == NSAlertDefaultReturn) {
        [[NSApplication sharedApplication] terminate:self];
        // Respond to applicationWillTerminate for cleanup
    }
}
```

Note

In [Figure 9.7](#), you see the window that is displayed when this application is run. By removing this window and setting the `NSBGOnly` entry in the `Info.plist` entries from [Figure 9.6](#) to a value of `1`, you can make the application "almost silent." By making these few changes, you can launch the application at login and simply have the menu appear with no other signs of the application running. It will not even appear in the Dock.

Try This

Here are a few things to try to implement using the MyStatusItem project as a start.

Remove the window and make it a background only application by setting the `NSBGOnlyInfo.plist` entry to `1`. Then, alter the first two menu items to display useful information, such as the My IP item does.

Try adding a menu item that displays a window that allows you to paste in the URL of an image file on the Internet. Load the image file and display it in a window with the height and width displayed as well as the size in bytes. Save the image to a file on the desktop.

Conclusion

The ability to create status items really opens the door for your programming ideas. Keep your eyes peeled for the publicizing of the `NSMenuExtra` API should it ever become so and consider using it instead of `NSStatusItem` for future development. Moreover, be sure to keep in mind the rules mentioned in the beginning of this chapter.

Chapter 10. Screen Effects

"Somebody save me."

—Remy Zero

Screensavers have been around since the first screen needed saving. Today, however, they are used more for fun than necessity. Back in the day of monochrome monitors, if you left a command prompt (or a menu bar) on the screen for a long period of time, you would suffer from the dreaded burn-in. The image on the screen would be permanently burned into the monitor. Screensavers would blank the screen and display moving objects so that you would still know the computer was running, but it wouldn't suffer from this horrible fate.

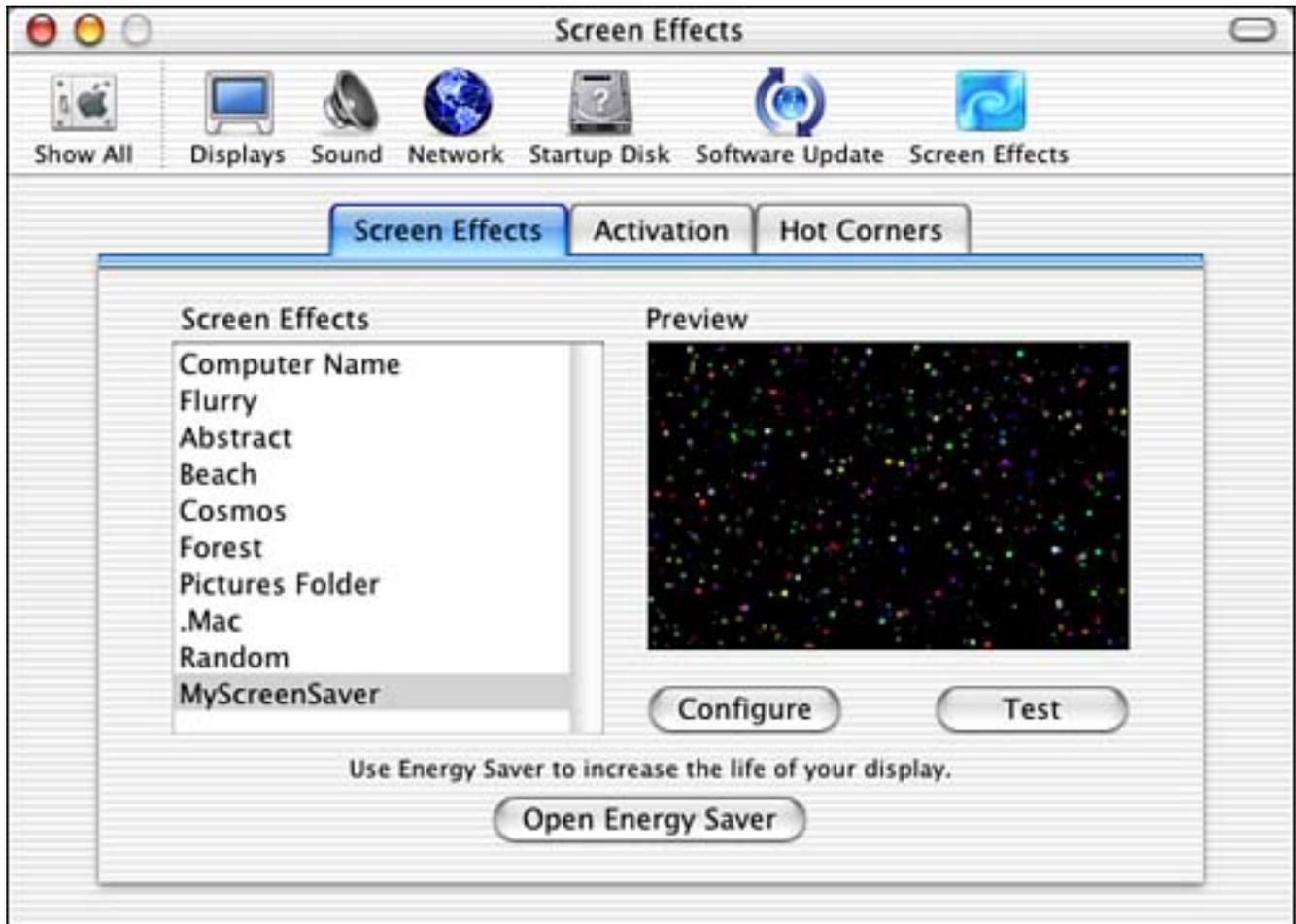
In the early days of the Macintosh, two popular screensavers came to life: Pyro, which made great looking fireworks and After Dark, which started the screensaver module revolution. After Dark was solely responsible for Flying Toasters and other such impossibilities. As display technology advanced and screensavers became more entertainment oriented, Apple decided to start calling them *screen effects*.

Note

Throughout this chapter, you will see some references to screensavers and some to screen effects. Since the name change with OS 10.2, not all parts of the OS have caught up. They are called different things in different places, at least for the time being. Eventually, this problem will fix itself.

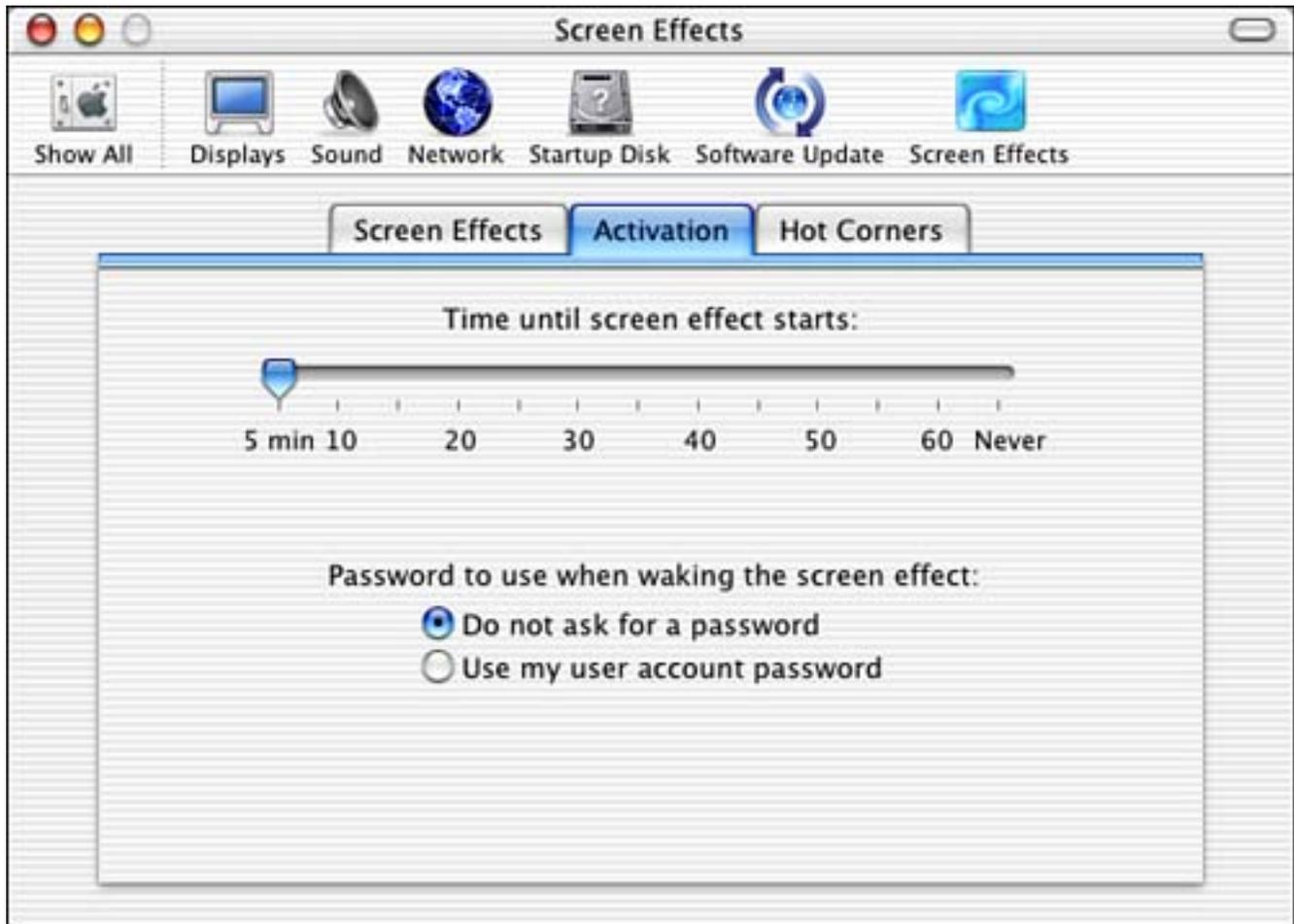
[Figure 10.1](#) shows the main Screen Effects tab of the Screen Effects preference pane in the System Preferences application. This tab allows you to choose which screen effect you would like to use for your screen. You can configure any available options of the screen effect from this tab and test it. The small Preview area shows you what the effect looks like with its current settings. Note that the screen effects in the list are read from the various **Screen Savers** directories. During development, I install mine in `~/Library/Screen Savers`, but the root `Library/Screen Savers` is also another popular spot. Most of Apple's screen effects are stored in `/System/Library/Screen Savers`.

Figure 10.1. The Screen Effects preference pane's Screen Effects tab in the System Preferences application.



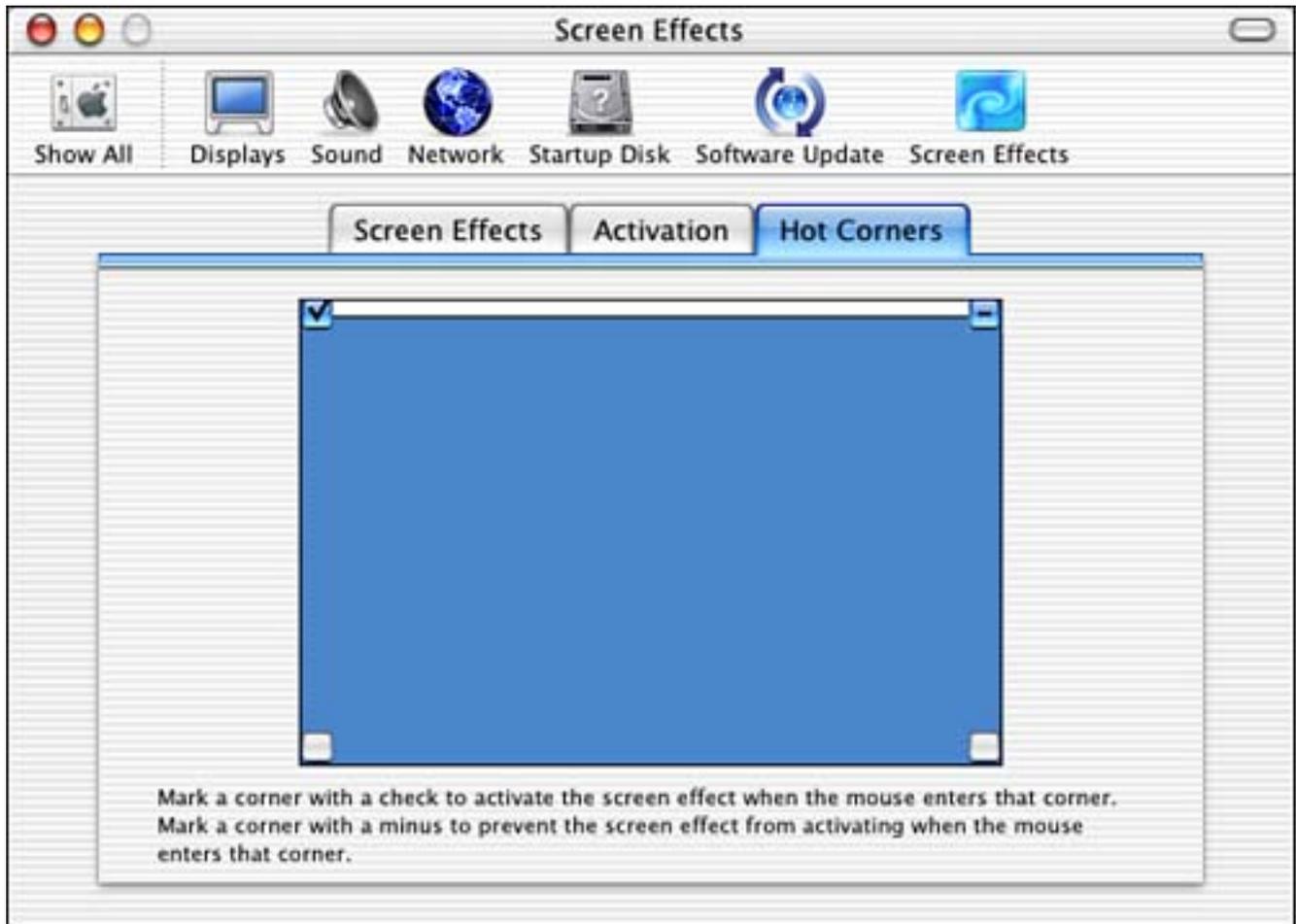
The Activation tab, shown in [Figure 10.2](#), allows you to set the time of inactivity required before the screen effect starts. You can also set a security option of whether to ask for the password when the computer wakes from a screen effect. These settings are global to all screen effects.

Figure 10.2. The Screen Effects preference pane's Activation tab in the System Preferences application.



[Figure 10.3](#) shows the Hot Corners tab where you can set the "Make Active Now" and "Never Make Active" corners. If you move the mouse into a corner with a check mark, the screen effect will come on immediately. If you move the mouse into a corner with a minus sign, the screen effect will never come on, effectively disabling the time setting in the Activation tab. These settings are also global to all screen effects.

Figure 10.3. The Screen Effects preference pane's Hot Corners tab in the System Preferences application.



Let's explore the screen effect we will be discussing in this chapter.

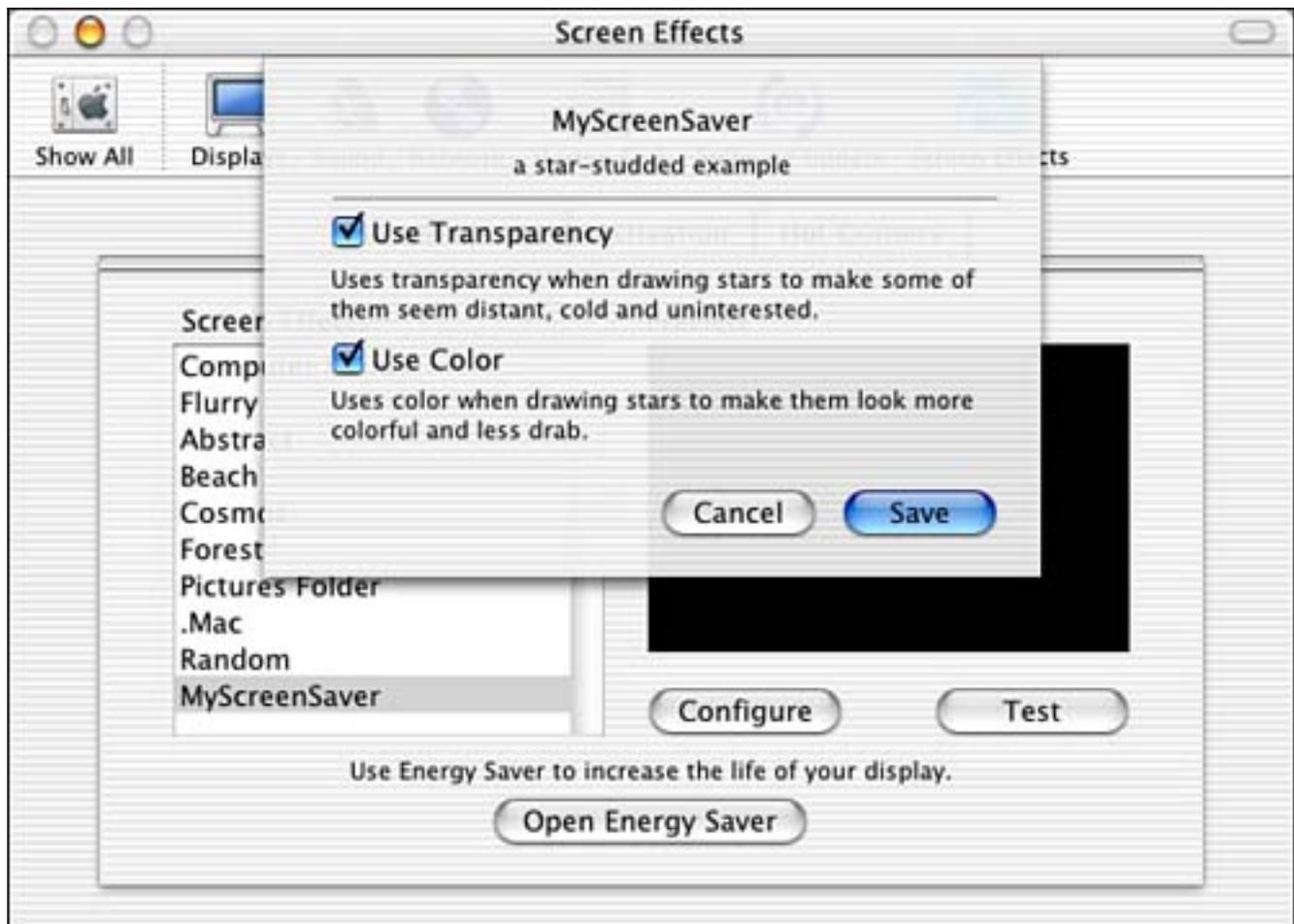
[\[Team LiB \]](#)

The Result

[Figure 10.1](#) shows the result of the project examined in this chapter. MyScreenSaver, which was written just before Apple started calling them screen effects, draws the representative particles of the outer reaches of deep space—also known as little colored dots.

[Figure 10.4](#) shows the results of pressing the Configure button on the Screen Effects tab. Note that two options are offered; Use Transparency and Use Color. Enabling Use Transparency draws the stars with a transparent effect, via the alpha channel, as you will see in the source code. Enabling Use Color draws the stars using color as opposed to grayscale only.

Figure 10.4. The MyScreenSaver screen effect configure sheet in the System Preferences application.



Note

What is an alpha channel? When you choose a color to draw, there are many ways to do so. The most popular way is to choose a red component, a green component, and a blue component. The additive effect of these choices creates any color in the spectrum. However, a fourth component is known as the alpha component or alpha channel. The alpha channel provides transparency to the colors so that you can see through them.

An alpha channel value between 0 and 1 creates the color as if it were stained glass—you can see through it in varying amounts depending on the value. An alpha channel value of 1 creates the color as if it were paint on wood—you can't see through it at all. An alpha channel value of 0 creates a color that is entirely transparent—like air; whatever other color components it might have will not be visible to mere mortals.

Let's look at the project file for MyScreenSaver.

[\[Team LiB \]](#)



The Project

[Figure 10.5](#) shows the MyScreenSaver project in Project Builder. The project was started by using the Screen Saver template in the New Project Assistant as shown in [Figure 10.6](#).

Figure 10.5. The MyScreenSaver project.

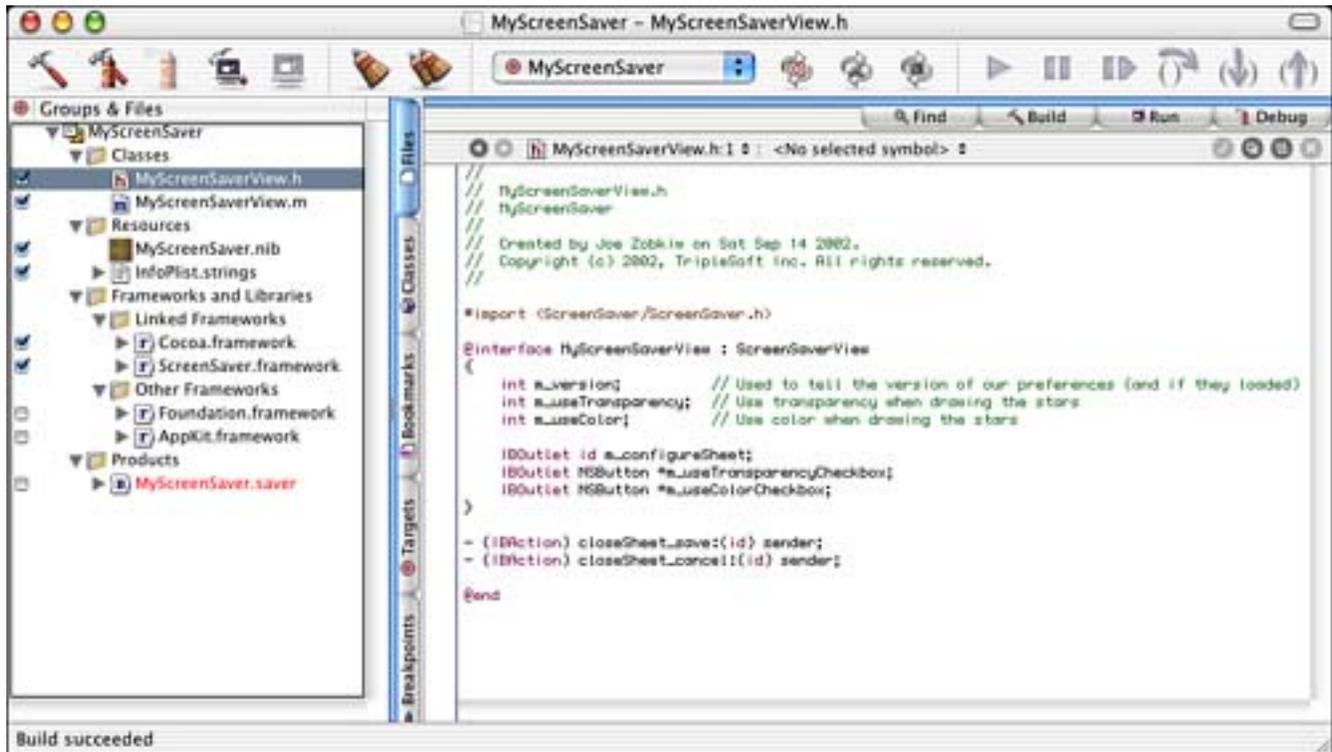
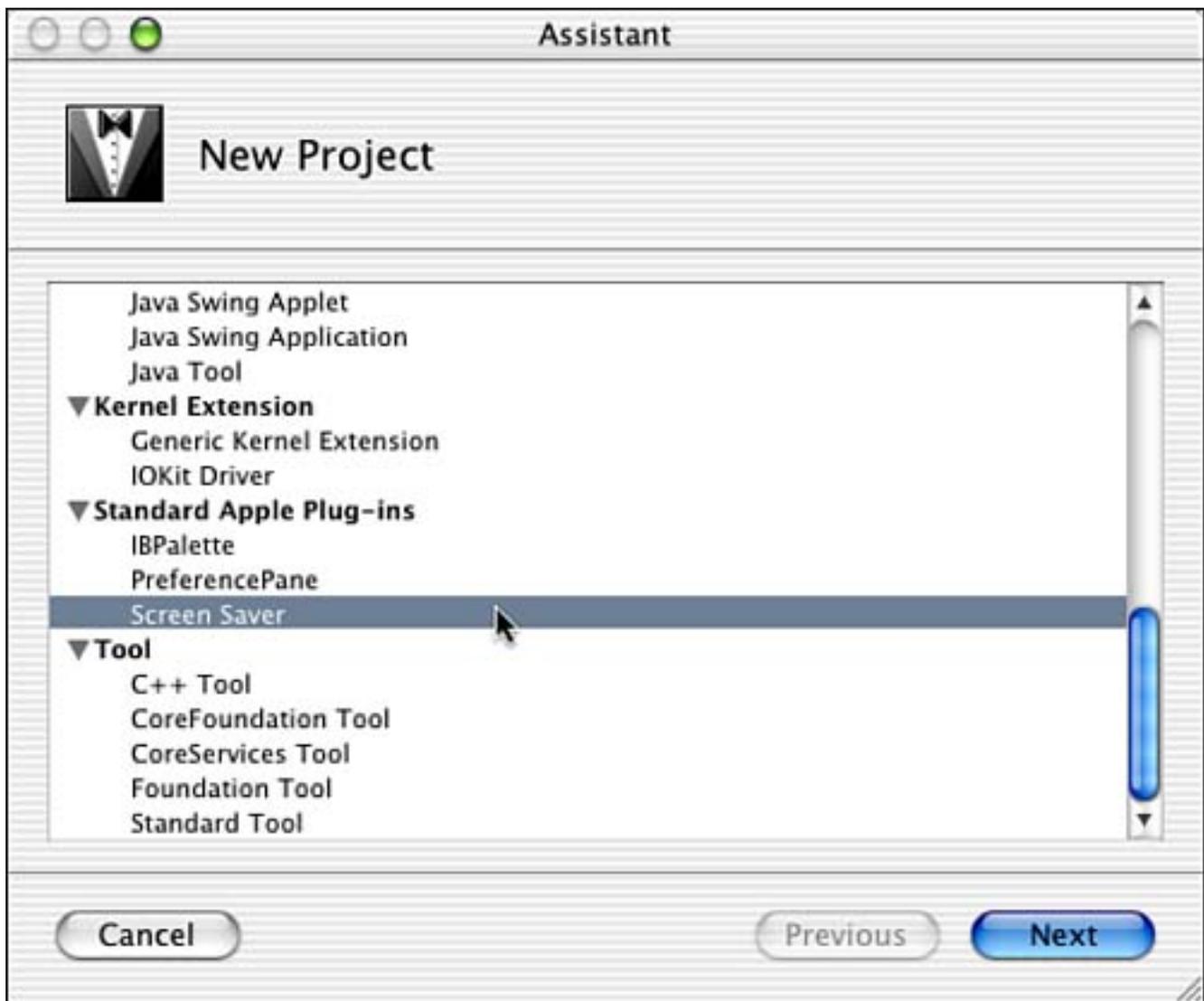


Figure 10.6. Selecting the Screen Saver template in the New Project Assistant.

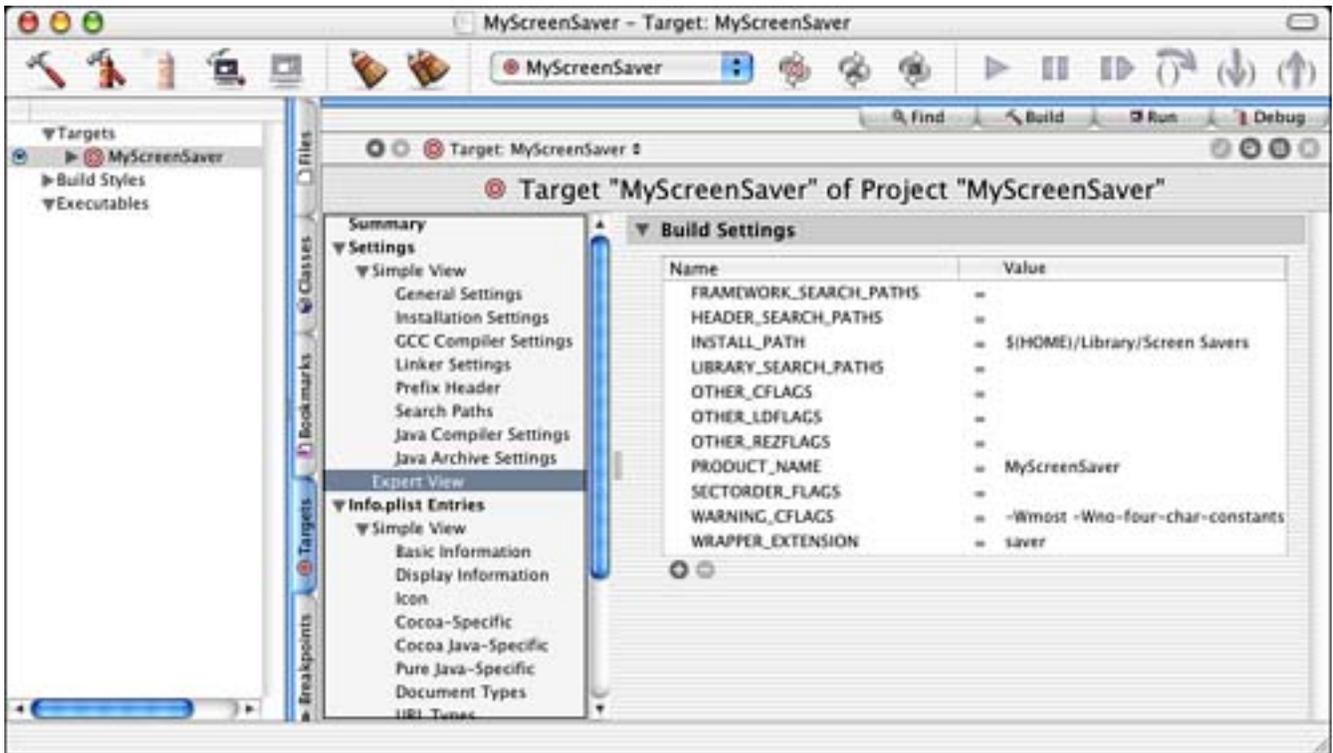


The project itself has only the `MyScreenSaverView.m` and `MyScreenSaverView.h` files to contend with, as well as the `MyScreenSaver.nib` file. If your screen effect doesn't have any configuration settings, you won't need to add a nib file to your project.

The Project Settings

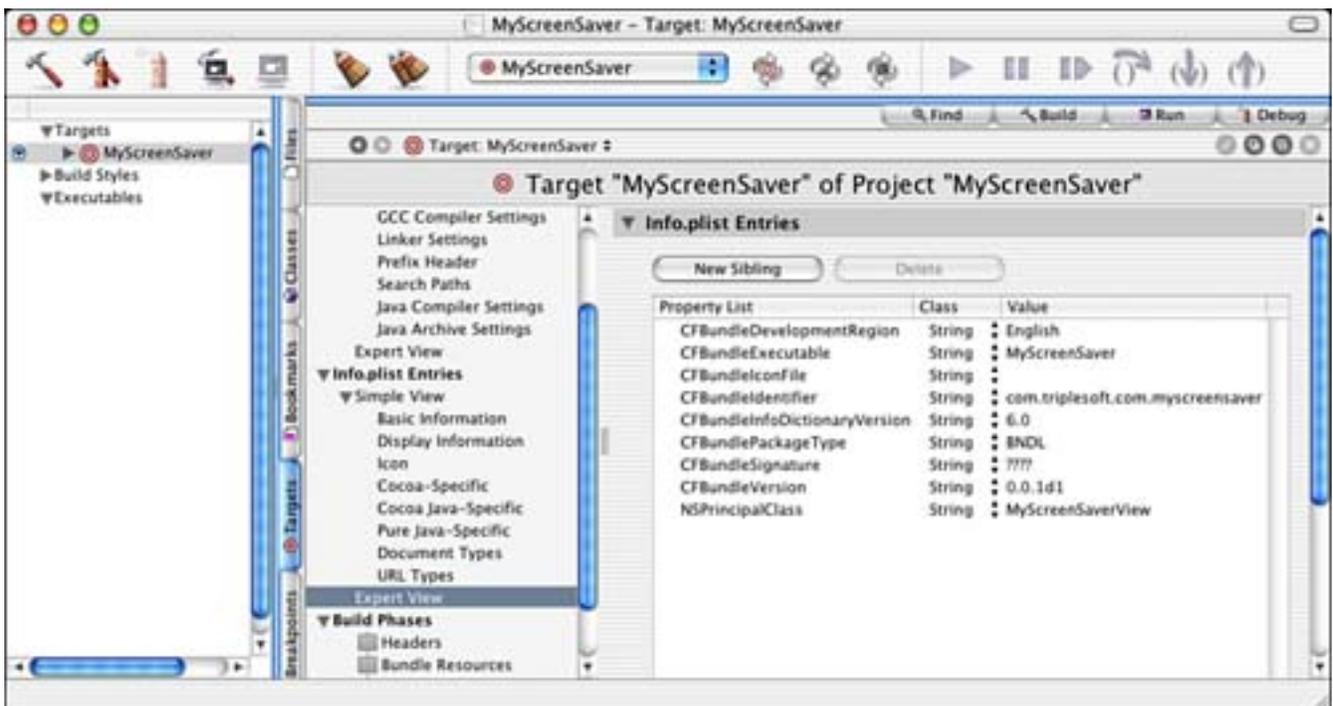
The Target settings in [Figure 10.7](#) are straightforward. Note that the `WRAPPER_EXTENSION` is set to `saver`. As of this writing, all screen effects must end in the extension `.saver`.

Figure 10.7. The MyScreenSaver Target settings.



The InfoPlist entries in [Figure 10.8](#) are also straightforward. Note that the `CFBundleIdentifier` is set to `com.triplesoft.myscreensaver`. The screen effect saves its configuration settings in this file, as you will see shortly. Note also that the `NSPrincipalClass` is `MyScreenSaverView`. I will discuss this class shortly as well.

Figure 10.8. The MyScreenSaver InfoPlist entries.



The Nib File

The `MyScreenSaver.nib` file is shown in Interface Builder in [Figure 10.9](#). Note that because we've set it as the File's Owner, we do not need to create a specific `MyScreenSaverView` instance. How handy is that? You can also see the `NSPanel` used to lay out the configuration sheet used in the screen effect (refer to [Figure 10.4](#)). The `IBOutlet`s in `MyScreenSaverView` are connected, as you would expect, to the `NSPanel` itself and the two check boxes within it. The Save and Cancel buttons are connected to two `IBAction`s that you will see in the source code. Note that we do not need an instance of `MyScreenSaverView` although the `MyScreenSaver.nib` file knows about it, as shown in [Figure 10.10](#).

Figure 10.9. The `MyScreenSaver` `MyScreenSaver.nib` file.

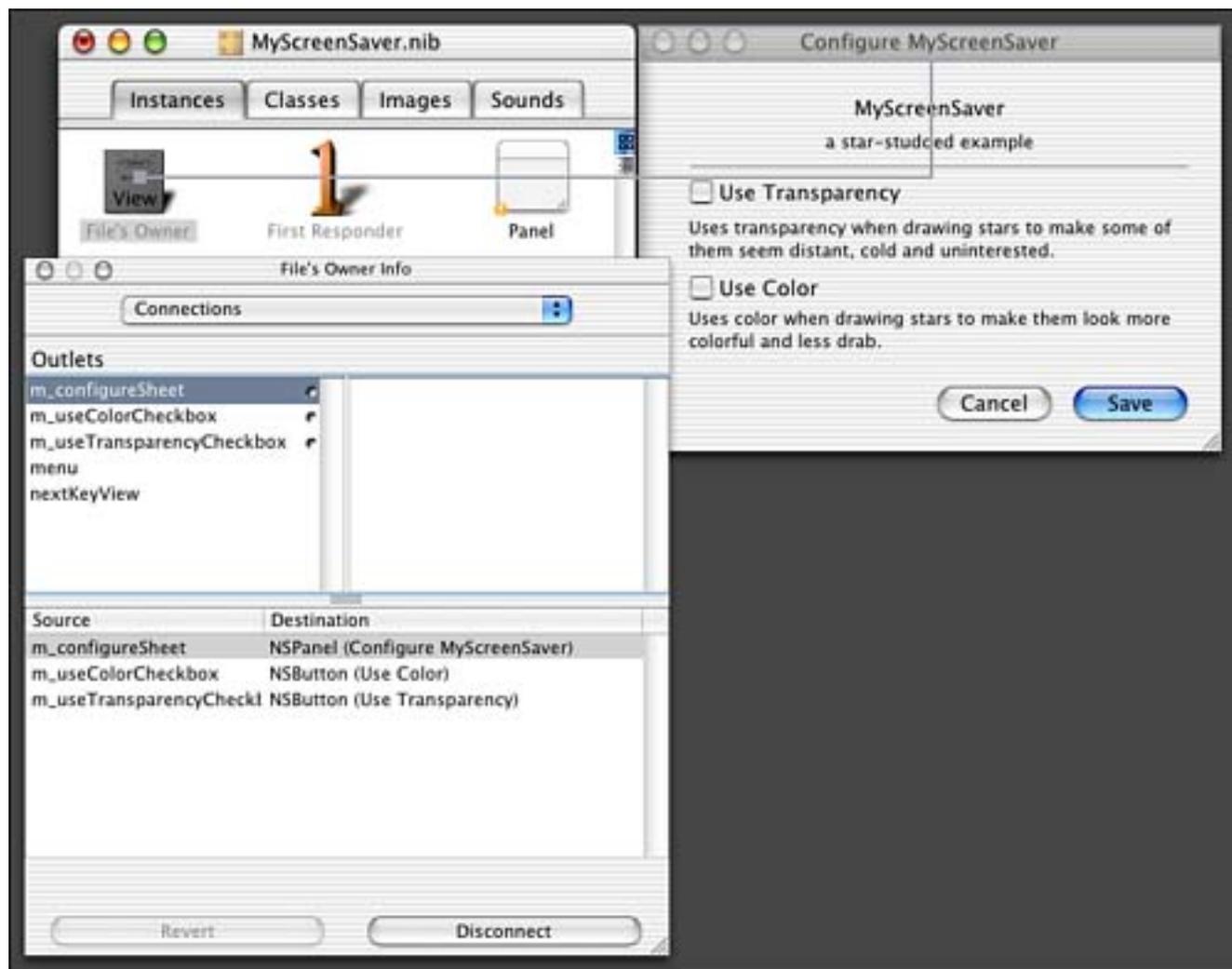
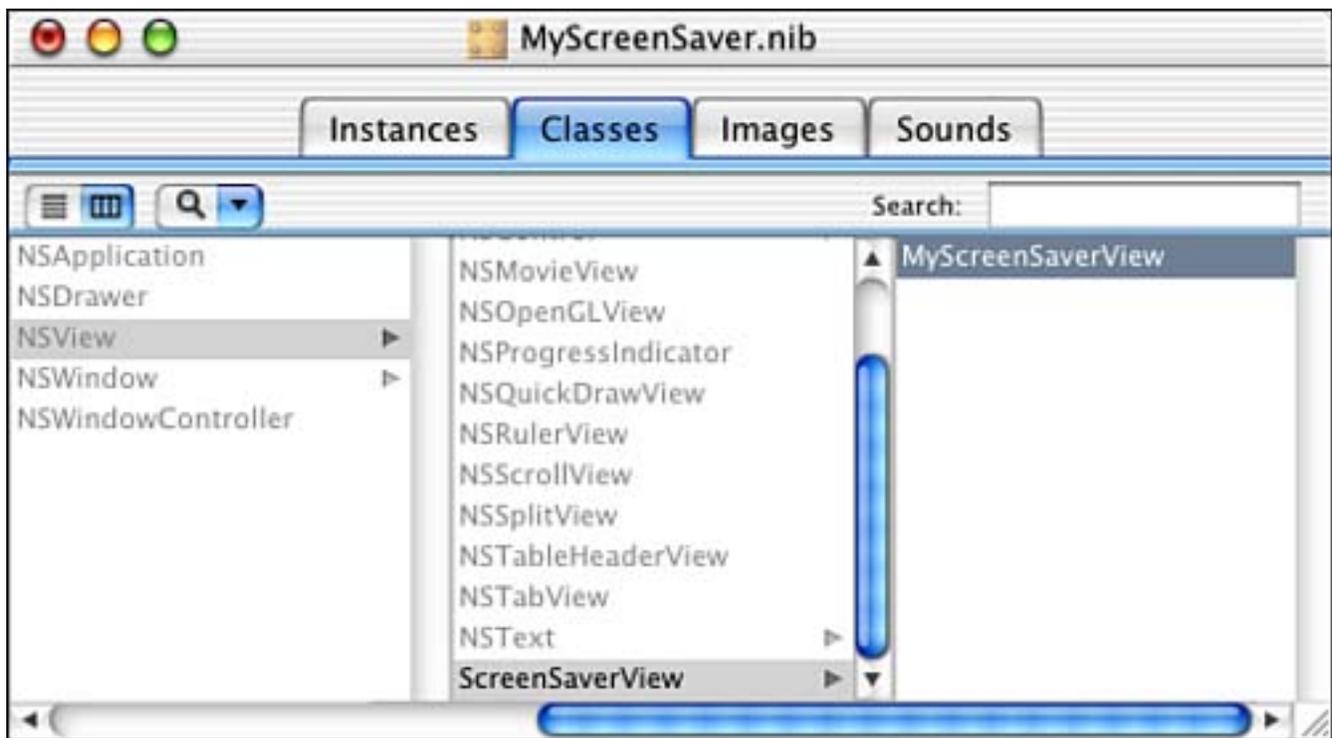


Figure 10.10. The `MyScreenSaver` `MyScreenSaver.nib` file `Classes` tab.



Let's look at how the code works to bring this all together.

The Source Code

The `MyScreenSaverView.m` and `MyScreenSaverView.h` files contain all the source code for our screen effect. `MyScreenSaverView` is a subclass of `ScreenSaverView`, which is a subclass of `NSView`. `ScreenSaverView` is defined in `ScreenSaver.h`. `MyScreenSaverView` is shown in [Listing 10.1](#).

Listing 10.1 `MyScreenSaverView` Interface in `MyScreenSaverView.h`

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

@interface MyScreenSaverView : ScreenSaverView
{
    int m_version;           // Used to tell the version of our preferences
                           // (and if they loaded)
    int m_useTransparency; // Use transparency when drawing the stars
    int m_useColor;        // Use color when drawing the stars

    IBOutlet id m_configureSheet;
    IBOutlet NSButton *m_useTransparencyCheckbox;
    IBOutlet NSButton *m_useColorCheckbox;
}

- (IBAction) closeSheet_save:(id) sender;
- (IBAction) closeSheet_cancel:(id) sender;

@end
```

Our subclass keeps track of a few instance variables that are managed during runtime to specify drawing

options, as well as the `IBOutlet`s for our configuration sheet and the two check boxes on it. We also have two `IBAction`s—one for the Save button and one for the Cancel button.

[Listing 10.2](#) contains the `-initWithFrame:isPreview:` method of our `MyScreenSaverView` class. This method is called to initialize the screen effect. After calling through to the super class, we load the default configuration settings for our bundle (`com.triplesoft.myscreensaver`) by calling the `ScreenSaverDefaults` class method `+defaultsForModuleWithName: .` `ScreenSaverDefaults` is a subclass of `NSUserDefaults`.

Listing 10.2 `MyScreenSaverView -initWithFrame:isPreview:` Method in `MyScreenSaverView.m`

```
- (id) initWithFrame: (NSRect) frame isPreview: (BOOL) isPreview
{
    // Initialize our super - the ScreenSaverView
    self = [super initWithFrame: frame isPreview: isPreview];

    if (self) { // If all went well...

        // Load the defaults for this screensaver
        ScreenSaverDefaults *defaults =
            [ScreenSaverDefaults defaultsForModuleWithName: kBundleID];

        // Try to load the version information to see if we have any
        // saved settings
        m_version = [defaults floatForKey:@"version"];
        if (!m_version) {
            // No saved setting, define our defaults
            m_version = 1;
            m_useTransparency = YES;
            m_useColor = YES;

            // Now write out the defaults to the defaults database
            [defaults setInteger:m_version forKey:@"version"];
            [defaults setInteger:m_useTransparency forKey:@"useTransparency"];
            [defaults setInteger:m_useColor forKey:@"useColor"];

            // And synchronize
            [defaults synchronize];
        }

        // Now, although this is overkill if version == 0, we load the
        // defaults from the defaults database into our member variables
        m_useTransparency = [defaults integerForKey:@"useTransparency"];
        m_useColor = [defaults integerForKey:@"useColor"];

        // And set the animation time interval (how often animateOneFrame is
        // called)
        [self setAnimationTimeInterval:1/30.0];
    }

    return self;
}
```

Note

Many professional screen effect module writers tend to get sick of defining something like `kBundleID` as used in [Listing 10.2](#). Instead, they use a dynamic method to find the bundle identifier as follows:

```
[[NSBundle bundleForClass:[self class]] bundleIdentifier]
```

This mechanism will always return the proper bundle identifier and allow you to more easily copy and paste your code from project to project.

Note

Why use `ScreenSaverDefaults` instead of `NSUserDefaults` or `CFPreferences`?

`NSUserDefaults` is great for getting and setting preferences for a specific application, such as RadarWatcherX. `NSUserDefaults` isn't as good for reading arbitrary default domains, nor is it good for per-user, per-machine settings such as screen effect settings. Because screen effects can be loaded into a variety of applications (system preferences, third-party programs, and so on), using an application-specific settings mechanism such as `NSUserDefaults` won't get you very far.

The `CFPreferences` API is powerful, but a little cumbersome for people who would rather do things in Objective C. Also, screen effects authors who use `CFPreferences` need to know specifically how their defaults should be stored. I thought that screen effects were supposed to be fun!

The `ScreenSaverDefaults` API gives screen effects authors a way to store per-user, per-machine settings without having to fiddle about with `CFPreferences`. It can be used to safely retrieve module settings no matter which application loaded the module. It also encourages people to store defaults consistently (that is, per user, per machine).

We then attempt to load the `m_version` instance variable using the `ScreenSaverDefaults` method `-floatForKey:`. If the value exists for this key, we know that we have saved settings. If the value does not exist, this is the first time the screen effect has been run. In that case, we initialize our values using the `-setInteger:forKey:` method of `ScreenSaveDefaults` and `-synchronize` them as well.

Note

At this point, you might consider using `NSUserDefaults`'s `registerDefaults:` method. The `registerDefaults:` mechanism allows you to say "If the user hasn't got a setting for XYZ, use this setting by default." This can do away with the need to commit the preference file in the way we are doing it here. This saves the creation of a preference file in the case in which a user simply looks at the screen effect, doesn't make any changes, and then throws it in the trash.

Now that we have the defaults either loaded or initialized, we can use the `-integerForKey:` method to get their values. In theory, we only have to do this if we loaded them; but to test the save and load of the defaults, we chose to do it this way during development. We could have easily added an `else` statement above these two lines of code.

Last, we set the animation interval timer by calling the `-setAnimationTimeInterval:` method of `MyScreenSaverView`. This determines how often the `MyScreenSaverView`'s `-animateOneFrame` method will be called while the screen effect is running. The default rate is 1/30th of a second, but you can change it to whatever you like.

Note

Did you know that you could change the frame rate of a screen effect while it's running?

Say, for example, that you have a slide show screen effect. Your user forgot to install pictures in your Pictures directory. Instead of blinking "You forgot to install your picture files!" 30 times a second, you can display it once and turn off the animation timer by using the undocumented trick of setting the animation rate to a value of less than 0.

Or, say that you've written some kind of basic computer-name display screen effect. Rather than displaying the name 30 times per second, you might want to display it once every 2 minutes.

Last, if you are an OpenGL speed freak, you might consider running the screen effect as fast as it will possibly go. Simply set the animate rate to 0 and order another battery for your laptop.

[Listing 10.3](#) shows the `-animateOneFrame` method. This method is called periodically based on the value passed to the `-setAnimationTimeInterval:` method. The first thing we do in this method is calculate the size of the star we will be drawing between 1 and 3 pixels in diameter by calling the `SSRandomIntBetween` function.

Listing 10.3 `MyScreenSaverView -animateOneFrame` Method in `MyScreenSaverView.m`

```
- (void) animateOneFrame
{
    NSRect    rect;
    NSColor   *color;
    int       x;
    float     r;

    // Calculate the star size
    x = SSRandomIntBetween(1, 3);

    // Make a rectangle at a random location the size of the star
    rect.origin = SSRandomPointForSizeWithinRect(NSMakeSize(x, x), [self bounds]);
    rect.size = NSMakeSize(x, x);
}
```

```

// Calculate a random color (or gray), with or without transparency,
// depending on the user preference
r = SSRandomFloatBetween(0.0, 1.0);
    // use red
color = [NSColor colorWithCalibratedRed:r
        // use new color or same as red if not using color
        green:m_useColor ? SSRandomFloatBetween(0.0, 1.0) : r
        // use new color or same as red if not using color
        blue:m_useColor ? SSRandomFloatBetween(0.0, 1.0) : r
        // use transparency if the user says so
        alpha:m_useTransparency ? SSRandomFloatBetween(0.0, 1.0) : 1.0];
[color set];

// Draw the star
[[NSBezierPath bezierPathWithOvalInRect:rect] fill];

return;
}

```

Next, we call `SSRandomPointForSizeWithInRect` to create a rectangle of the determined size at a random location within the bounds of the rectangle that the screen effect will be drawn in. This will either be the same rectangle as the entire screen or that of the preview pane as shown in [Figure 10.1](#).

Note

What's with these random number functions?

Unlike most other system-level software, screen effects tend to compute many random numbers. Rather than making users remember the right way to compute a random number using the UNIX `random()` or POSIX `rand()` API, Apple created a few inline functions for doing exactly this.

"But what about the random seed," you ask? The screen effect engine initializes the random seed once before loading screen effects, so your module doesn't need to worry about that itself. In fact, modules that initialize the random seed in their `init()` routines aren't very interesting on multi-monitor machines—because each monitor might start with the same image!

We then calculate our random `NSColor`, taking into account the transparency and the color options. The `-colorWithCalibratedRed:green:blue:alpha:` method is one you will find yourself using over and over again when doing simple drawing. Note that all the values are floats between `0.0` and `1.0`. All values of `0.0` would create a completely clear color, just like `+ [NSColor clearColor]`—remember the alpha channel. All values of `1.0` would create white. We then `-set` the `NSColor` and draw our `NSBezierPath` by calling the `-bezierPathWithOvalInRect:` and `-fill` methods. Note that the `-bezierPathWithOvalInRect:` method takes into account anti-aliasing and draws a very nice, smooth oval.

Note

When you look at the source code of this project, you will see that you can use a `-drawRect:` method to draw your screen effect. Depending on your needs, this might or might not be a better approach. See the "[Try This](#)" section at the end of this chapter for a pointer on using `-drawRect:`.

[Listing 10.4](#) is a simple one. The `-hasConfigureSheet` method simply returns `YES` or `NO` depending on whether our screen effect has a configure sheet. Because we do, we must override this method and return `YES`.

Listing 10.4 `MyScreenSaverView -hasConfigureSheet` Method in `MyScreenSaverView.m`

```
- (BOOL) hasConfigureSheet
{
    return YES;
}
```

[Listing 10.5](#) contains the `-configureSheet` method that is responsible for returning the `NSWindow` that is our configure sheet. You might recall that our configure sheet is stored in the `MyScreenSaver.nib` file from [Figure 10.9](#). Because we store the configure sheet as an instance variable, we only need to load it once. Therefore, if it is `NULL`, we call the `NSBundle` class method `+loadNibNamed:owner:` method to load the `MyScreenSaver.nib` file. This automatically loads the configure sheet and sets the `m_configureSheet` instance variable (`IBOutlet`) to point to the sheet itself. The check box instance variables are also set to point to their user interface equivalents during the load. This makes it simple to set the state of the check boxes to the current values of the screen effect settings and return the sheet. Note also that the Save and Cancel buttons are linked to their actions during this load as well. The sheet is then displayed to the user as shown in [Figure 10.4](#).

Listing 10.5 `MyScreenSaverView -configureSheet` Method in `MyScreenSaverView.m`

```
// Display the configuration sheet for the user to choose their settings
- (NSWindow*) configureSheet
{
    // If we have yet to load our configure sheet,
    // load the nib named MyScreenSaver.nib
    if (!m_configureSheet)
        [NSBundle loadNibNamed:@"MyScreenSaver" owner:self];

    // Set the state of our UI components
    [m_useTransparencyCheckbox setState:m_useTransparency];
    [m_useColorCheckbox setState:m_useColor];

    return m_configureSheet;
}
```

After the user alters the settings in the sheet and he clicks the Save button, the `IBAction` method in [Listing 10.6](#) is called. This method loads the current default values for the screen effect, saves the current state of the check boxes to our instance variables, writes the new values to the defaults, synchronizes them, and calls `NSApp's +endSheet:` class method to close the sheet.

Listing 10.6 MyScreenSaverView -closeSheet_save: Method in MyScreenSaverView.m

```
// The user clicked the SAVE button in the configuration sheet
- (IBAction) closeSheet_save:(id) sender
{
    // Get our defaults
    ScreenSaverDefaults *defaults = [ScreenSaverDefaults
        defaultsForModuleWithName:kBundleID];

    // Save the state of our UI components
    m_useTransparency = [m_useTransparencyCheckbox state];
    m_useColor = [m_useColorCheckbox state];

    // Write them to the defaults database
    [defaults setInteger:m_useTransparency forKey:@"useTransparency"];
    [defaults setInteger:m_useColor forKey:@"useColor"];

    // Synchronize
    [defaults synchronize];

    // The sheet has ended, go in peace
    [NSApp endSheet:m_configureSheet];
}
```

If the user had pressed the Cancel button instead of Save, there would've been nothing to do. The code in [Listing 10.7](#) is executed in this case and merely closes the sheet by calling `NSApp's +endSheet:` class method.

Listing 10.7 MyScreenSaverView -closeSheet_cancel: Method in MyScreenSaverView.m

```
// The user clicked the CANCEL button in the configuration sheet
- (IBAction) closeSheet_cancel:(id) sender
{
    // Nothing to do! The sheet has ended, go in peace
    [NSApp endSheet:m_configureSheet];
}
```

Note

If you poke around the various Screen Saver folders, you will see some screen effects that end in `.slideSaver` in OS X 10.1 or later. These screen effects have no associated source code and are nothing more than a bundle filled with `.jpg` files in their Resources folder. Find one on your drive and take it apart, and then consider creating your own with pictures of your pet, kids, or yourself.

It wasn't that bad, huh? Ready to try some modifications on your own?

Try This

You can do so many cool things in a screen effect. Here are some to try using this project as a starting point.

Add a control to set a maximum number of stars. When this maximum number is reached, don't create any more than that number. You can choose to clear the screen and start over when you reach the maximum or keep track of the stars as you create them and redraw the same ones in a different color after the limit is reached.

Search out an image of the moon and draw it in a random location before you begin to draw the stars. You can look back at previous chapters to see how to load an image either from a resource or via the Internet and then draw it into a view. Notice how the transparency setting allows some of the stars to be seen through. Make the stars that appear over the moon image draw without transparency even if the option is set.

Try doing all of your drawing in `-drawRect:` and invalidate the `ScreenSaverView` in `-animateOneFrame` using `[self setNeedsDisplay:YES];`.

Try using a separate `NSWindowController` object to manage the configure sheet. Use a delegate message or notification to communicate to the screen effect.

Try improving performance by drawing more than one star at a time. Experiment with other drawing methods.

Conclusion

Whether you call them screensavers or screen effects, they are exciting pieces of code to implement. The nice thing about them is that they are self-contained. Once you put together a template project (such as the one outlined here), you can let your imagination run wild and implement any type of crazy animation you desire. There are plenty of great examples of screen effects to use as examples—from awesome lifelike flags waving in the wind to three-dimensional vortexes being twisted and turned. Screen effects are a great place to explore the power of OpenGL, the high-performance 2D/3D graphics environment. For more information on OpenGL, visit <http://www.opengl.org/>.

Chapter 11. Color Pickers

"Yellow and blue make green."

—Basic scientific fact

I remember my first color computer; it was a Macintosh IIcx and was purchased with the help of my Uncle David. Moving from a Macintosh Plus to a Macintosh IIcx was an incredible experience. Not only did I have a faster computer that was truly expandable (via NuBus slots—remember those), but also I could finally program in color!

So here I am, programming in color, making icons, color tables, simple games, and utilities. The first thing I learned to do programmatically was how to let the user choose the color she wanted for any particular need. The Macintosh OS has had a system color picker since Color QuickDraw, now called the color panel. With OS X, it is better than ever although a bit light on documentation as of this writing.

[Figure 11.1](#) shows the system color panel with the Color Wheel color picker selected. To clarify, the color panel is the entire window that manages the individual color pickers. The color wheel is one such color picker—one of many such color pickers.

Figure 11.1. The standard system color panel showing the Color Wheel color picker.



The color panel is usually displayed in relation to a font choice or in a drawing program to choose the color of a paintbrush or other such tool. Once displayed, the color panel displays all available color pickers as icons along the top of its window. [Figure 11.1](#) shows the following color pickers: color wheel, color sliders, color palettes, image palettes, crayons, and MyColorPicker—the one we will examine in this chapter. Custom color pickers live in the `/Library/ColorPickers` folder, whereas system color pickers (the ones from Apple) usually live in the `/System/Library/ColorPickers` folder. Color picker names always end in `.colorPicker`.

An application programmer need not worry about how the user chooses to pick her color in the color panel. The application programmer merely asks the color panel for a color. The user can then move back and forth between color pickers to get just the right color. When all is said and done, the color panel returns the chosen color to the calling application. You can also easily limit the types of color panels that are displayed depending on the types of colors your application supports. For example, you choose to only show color pickers that support RGB or CMYK color modes. Refer to Apple's documentation on using color wells and color panels for more information on using these at the application level.

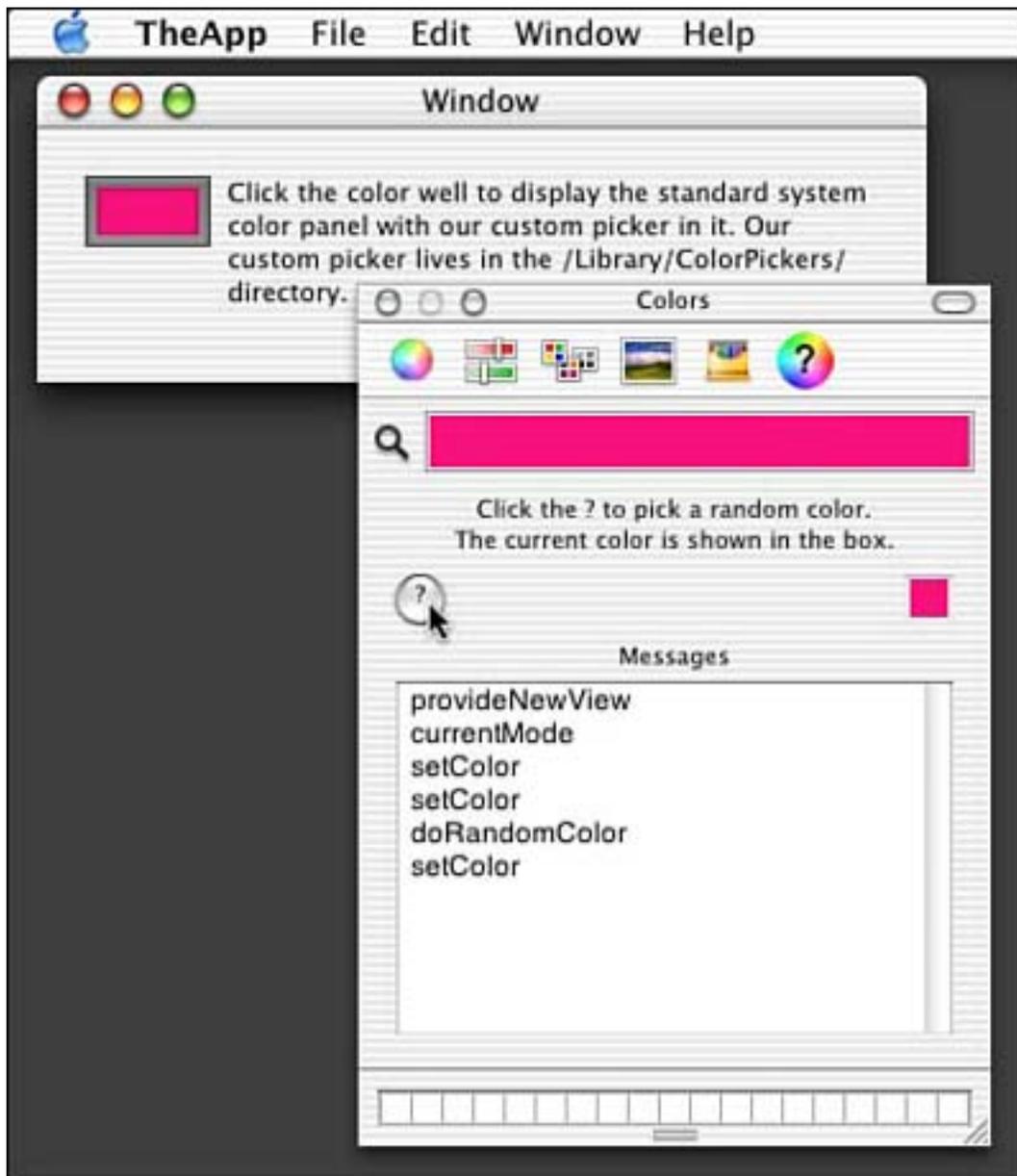
Let's look at the results of this chapter.

[\[Team LiB \]](#)

The Result

[Figure 11.2](#) shows MyColorPicker as the current color picker in the color panel. The window behind the color panel is the main window of our test application. This application is a simple Cocoa application that contains a window with a color well. There is no supporting code other than what was supplied by the Cocoa Application project template. The color well, when clicked, automatically displays the color panel, which displays the currently selected color in it. This application provides a quick and easy way to display the color panel during development.

Figure 11.2. The MyColorPicker color picker being used in our test application.



MyColorPicker is very much a test bed color picker. It does not provide any remarkable way to choose a color. What it does do, however, is track the color picker messages that it handles as the user interacts with it. It also provides a way to choose a random color by clicking the ? button and displays it in the small

square to the right of the window. The large color box across the top and the magnifying glass tool (just below the color picker icons) are provided by the color panel itself. This is also true for the color wells at the bottom of the window, which are used to store your favorite colors via drag and drop.

Let's look at the project.

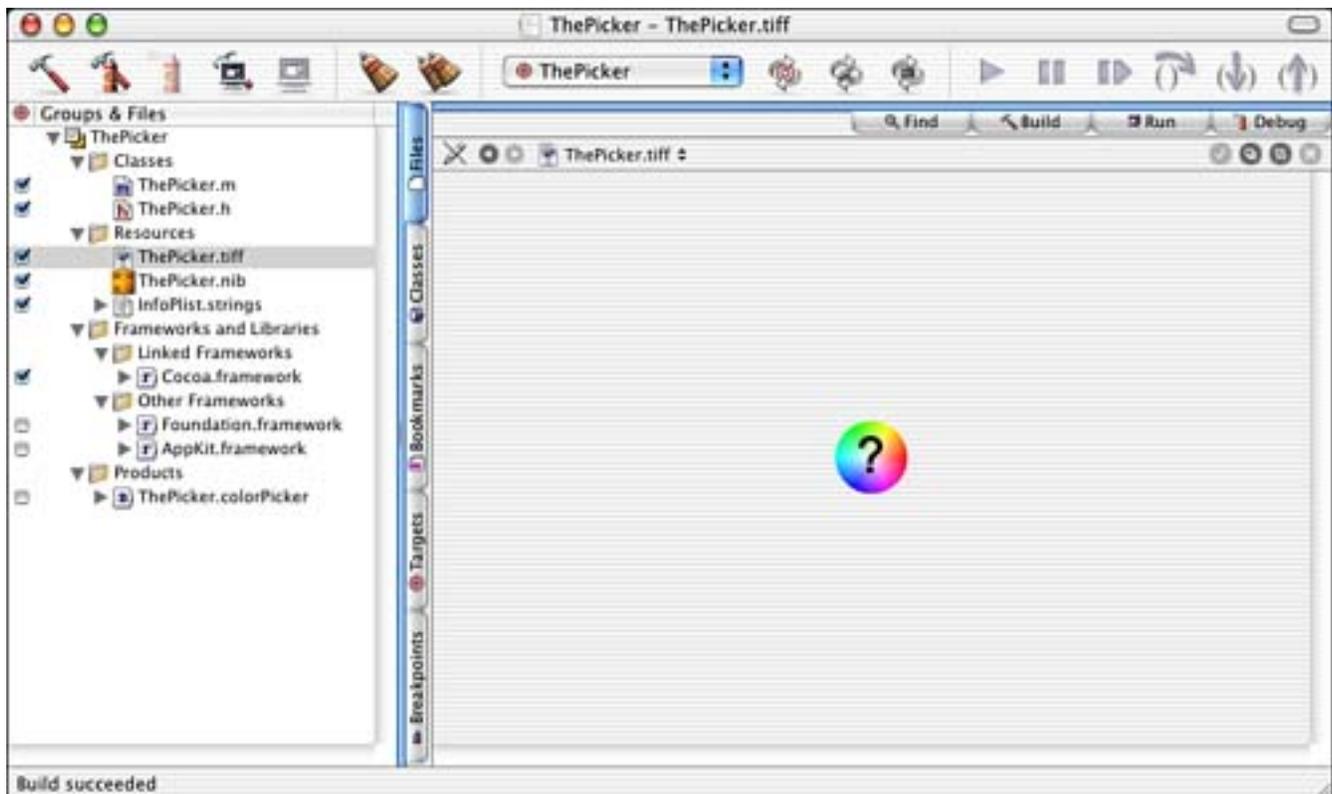
[\[Team LiB \]](#)



The Project

[Figure 11.3](#) shows the MyColorPicker project in Project Builder. The project was started using the Cocoa Bundle template in the New Project Assistant and then edited appropriately. Note that the project has only a few files. `ThePicker.m` and `ThePicker.h` contain all the code for the color picker itself. `ThePicker.tiff` contains the icon used to represent our color picker in the color panel. `ThePicker.nib` contains the user interface elements that make up our color picker.

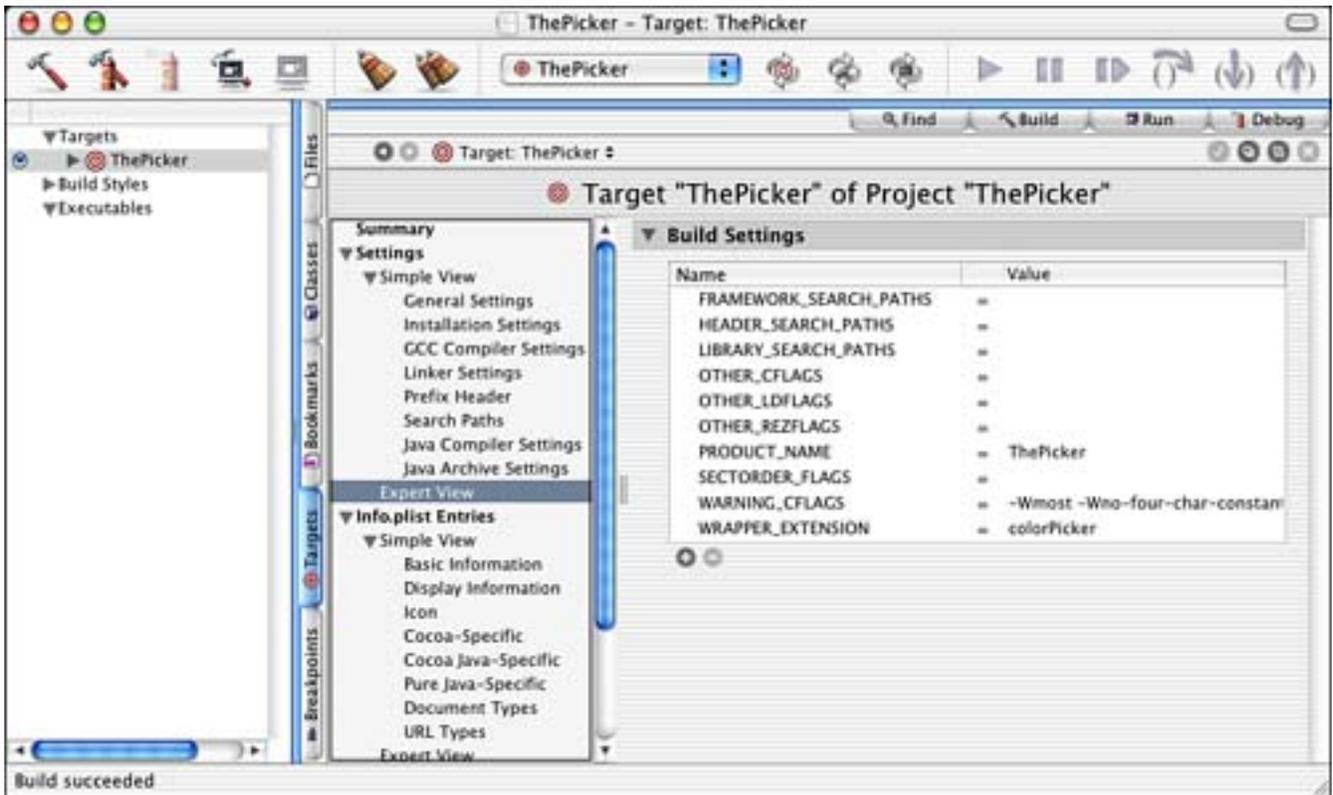
Figure 11.3. The MyColorPicker project.



The Project Settings

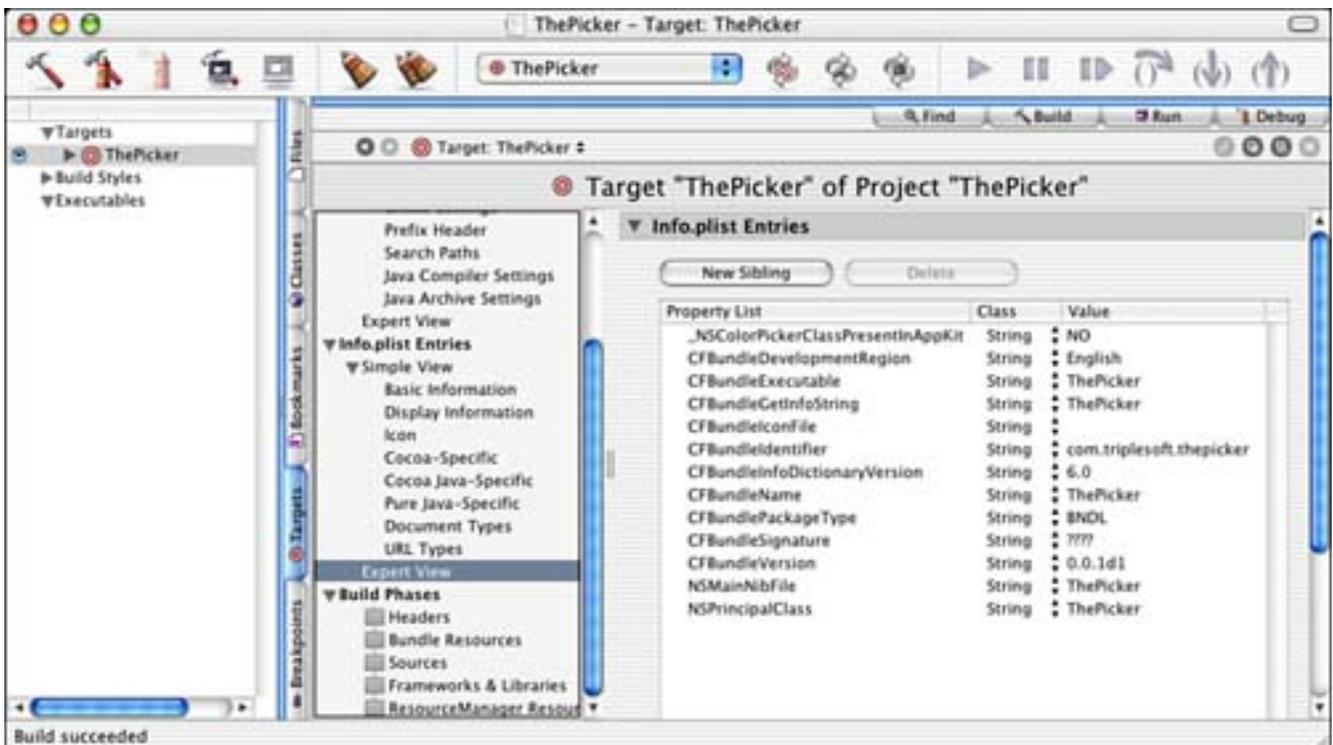
The Target settings in [Figure 11.4](#) are straightforward. Make note that the `WRAPPER_EXTENSION` is set to `colorPicker`. All color pickers must end in this suffix to be recognized as such. This is nothing new if you've been following along in previous chapters.

Figure 11.4. The MyColorPicker Target settings.



The InfoPlist entries in [Figure 11.5](#) bring a few new items to the table, however. Note the `_NSColorPickerClassPresentInAppKit` entry, which is used internally to the color picker implementation. Note also that the `NSMainNibFile` entry is `ThePicker`, which refers to `ThePicker.nib`. The `NSPrincipalClass` is also `ThePicker`, which is the name of our `NSColorPicker` subclass that we will see shortly.

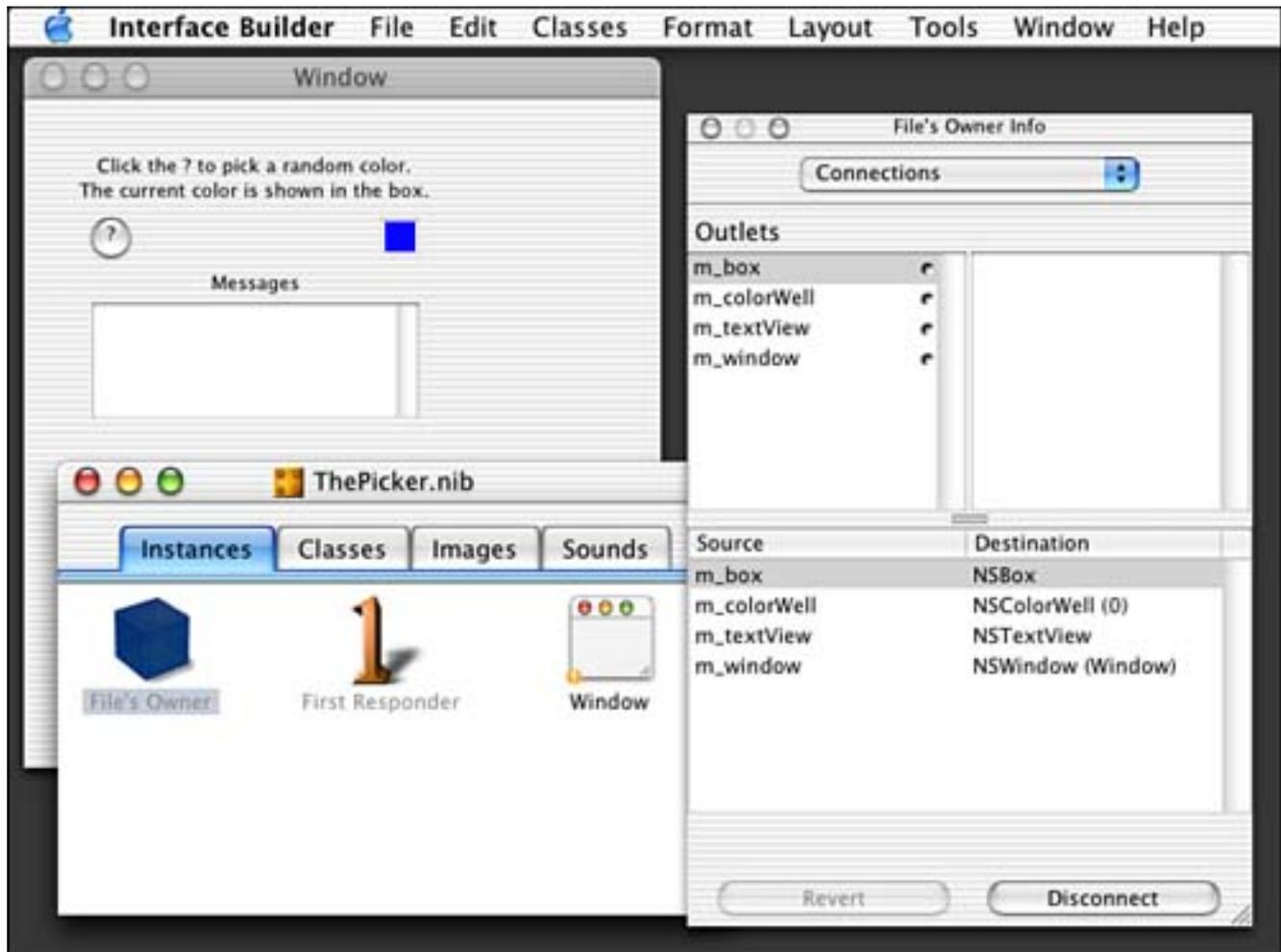
Figure 11.5. The MyColorPicker InfoPlist entries.



The Nib File

The `ThePicker.nib` file is shown in [Figure 11.6](#) as it appears in Interface Builder. The File's Owner is our `ThePicker` class and the items in our `NSWindow` are connected to its `IBOutlet`s accordingly. Although you cannot see it in this picture, the `?` button is also connected to an `IBAction` in `ThePicker`, `-doRandomColor:`.

Figure 11.6. The MyColorPicker `ThePicker.nib` file.



Note that the user interface layout, as shown in [Figure 11.7](#), is straightforward, but there is one key that you need to be aware of. The items in the user interface are encompassed by an `NSBox`, outlined in the picture. This box allows us to easily refer to a group of items in the user interface in one fell swoop and pass them to the color panel. The `NSBox` is the parent view of the other items; where the box goes, they go—always as a group. This technique can be used in a variety of programs—wherever groups of controls need to be managed, moved, or swapped in and out. The use of a tabs metaphor works similarly under the hood.

Figure 11.7. The MyColorPicker user interface layout in `ThePicker.nib` file



Note

You don't have to worry too much about the size of your `NSBox` and items. The color panel will resize itself to fit whatever items you have in the box.

In fact, there is another way to do this entirely without using `NSBox` at all.

Interface Builder now allows you to create windowless views, which are great for this type of thing. Simply drag a view (custom or otherwise) into your nib file's main window (as if you were dragging a window), and it will "just work."

Let's look at the source code.

The Source Code

I will admit it, once `MyColorPicker` got working, I looked at the pieces and it all seemed to fit together; however, getting to that point was a difficult task. There are practically no examples or good documentation for creating a color picker as of this writing. The only other color picker I found was a shareware program that did not have source code available. Trial and error and a few well-placed emails and postings got this project working.

[Listing 11.1](#) shows the interface for `ThePicker`. `ThePicker` is a subclass of `NSColorPicker` and implements the `NSColorPickerCustom` protocol. You might recall that a protocol is a way to create methods to be implemented without being related to or dependent on any specific class. `NSColorPicker` is an abstract super class that implements two such protocols: `NSColorPickerDefault` and `NSColorPickerCustom`. The `NSColorPickerDefault` protocol provides basic, default behavior for a color picker. The `NSColorPickerCustom` protocol provides custom, implementation-specific behavior. Because `NSColorPicker` already adopts the `NSColorPickerDefault` protocol and we are a subclass of `NSColorPicker`, we only need to additionally adopt the `NSColorPickerCustom` protocol.

Listing 11.1 ThePicker Interface in ThePicker.h

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

@interface ThePicker : NSColorPicker <NSColorPickerCustom>
{
    IBOutlet NSWindow *m_window;
    IBOutlet NSBox *m_box;
    IBOutlet NSTextView *m_textView;
    IBOutlet NSColorWell *m_colorWell;

    NSColor *m_color;
}

- (IBAction) doRandomColor: (id) sender;
- (void) logText: (NSString *) theString;

@end
```

Note that `ThePicker` contains some `IBOutlet`s to manage our user interface elements. It also tracks the current color being displayed as an `NSColor`. There is also an `IBAction` to handle the clicking of the `?` button to create a random color and a `-logText:` method to allow the tracking of the messages as they pass through.

`NSColorPickerDefault`

First let's look at the methods that are implemented as part of the `NSColorPickerDefault` protocol. Note that there are other `NSColorPickerDefault` protocol methods that we need not worry about in this project. You can find them documented in `NSColorPanel.h`.

[Listing 11.2](#) shows the `-initWithPickerMask:colorPanel:` method. This method is called to initialize our picker. It allows us to check the mode in which the color panel is being displayed. If we do not support the mode that the color panel requests, we do not initialize our super class. Also, we are not displayed as a potential color picker in this case. Because we only support RGB mode colors—the most common—we check for that in the `mask` parameter. The `owningColorPanel` parameter is the color panel itself. This is stored by the `NSColorPicker` super class and can be accessed later, as we will do, via `NSColorPicker`'s `-colorPanel` method. Note that we also initialize the random number seed so that when we choose a random color later on, it truly is random.

Listing 11.2 ThePicker -initWithPickerMask:colorPanel: Method in ThePicker.m

```

- (id) initWithPickerMask: (int) mask colorPanel: (NSColorPanel *) owningColorPanel
{
    if (mask & NSColorPanelRGBModeMask) { // we only support RGB mode
        [super initWithPickerMask: mask colorPanel: owningColorPanel];
    }
    srand(time(0)); // init random number seed
    return self;
}

```

[Listing 11.3](#), the `-provideNewButtonImage` method, is called when the color panel needs the image used to represent our color picker to the user. This is the `ThePicker.TIFF` file shown in [Figure 11.3](#). We simply allocate a new `NSImage` with the contents of the `TIFF` file. This method contains many nested references, but essentially—once we get the `NSBundle` based on our class—we can pull the image resource from it and use that to initialize the `NSImage`.

Listing 11.3 ThePicker -provideNewButtonImage Method in ThePicker.m

```

- (NSImage *) provideNewButtonImage
{
    return [[NSImage alloc] initWithContentsOfFile:
        [NSBundle bundleForClass: [self class]]
        pathForResource:@"ThePicker"];
}

```

Note

Note that these methods are called before our interface has been created. Given that, there are no calls to the `-logText:` method that you will see in later code.

NSColorPickingCustom

Now let's look at the methods that are implemented as part of the `NSColorPickingCustom` protocol. Note that there are other `NSColorPickingCustom` protocol methods that we need not worry about in this project. You can find them documented in `NSColorPanel.h`.

The `-setColor:` method in [Listing 11.4](#) is called whenever the color panel needs to tell our color picker that it needs to display a new color. For example, if the user uses the magnifying glass tool (shown in [Figure 11.2](#)) to choose a color from the screen, the color panel will send the `-setColor:` message to the currently displayed color picker. In our case, we `-retain` the new color, `-release` the old, and copy the new to our instance variable. We also set the color of the color well in our user interface (the small square on the right) to the new color. It is up to the color picker to take the new color and display it in whatever way is deemed appropriate for the user interface. For example, a color picker with sliders might set all the slider values to designate the values other red, green, and blue components of the newly chosen color.

Listing 11.4 ThePicker setColor Method in ThePicker.m

```

- (void) setColor: (NSColor *) color
{

```

```

    [self logText:@"setColor\n"];
    [color retain];
    [m_color release];
    m_color = color;
    [m_colorWell setColor:m_color];
}

```

The `-currentMode` method in [Listing 11.5](#) is used to report the mode that your color picker operates. This should be a unique value for your color picker. We defined `ThePickerMode` to be `100` elsewhere in the `ThePicker.m` file. You need to pick a unique number that is not the same as any of the standard color pickers defined in `NSColorPanel.h`. There is no way to register your number with Apple as far as I can tell. But it seems that color pickers aren't the most popular items to implement anyway, so you should be safe picking an obscure number. Pick the last three digits of your phone number plus 100 to be safe!

Listing 11.5 ThePicker -currentMode Method in ThePicker.m

```

- (int)currentMode
{
    [self logText:@"currentMode\n"];
    return ThePickerMode;
}

```

The `-supportsMode:` method in [Listing 11.6](#) is a corollary to `-currentMode`. You should return `YES` if you support the mode passed in the `mode` parameter; otherwise return `NO`.

Listing 11.6 ThePicker -supportsMode: Method in ThePicker.m

```

- (BOOL)supportsMode: (int) mode
{
    [self logText:@"supportsMode\n"];
    return (mode == ThePickerMode);
}

```

The `-provideNewView:` method in [Listing 11.7](#) is an important one. This method returns the `NSBox` that we discussed earlier in [Figure 11.7](#). Whenever the color panel needs the view to display for a color picker, it calls this method. If this is the first time the method has been called, we attempt to load the nib file and then simply return the `m_box` instance variable, which will be automatically initialized during the load. Otherwise, we just return the previously initialized `m_box`.

Listing 11.7 ThePicker -provideNewView: Method in ThePicker.m

```

- (NSView *)provideNewView: (BOOL) initialRequest
{
    if (initialRequest) {
        if ([NSBundle loadNibNamed:@"ThePicker" owner:self]) {
            [self logText:@"provideNewView\n"];
            return m_box;
        } else {
            NSBeep();
            NSRunAlertPanel(@"Error", @"Couldn't load nib.",
                @"OK", @"", @"");
        }
    }
}

```

```

    }
}
return m_box;
}

```

Note

Learn from my mistake! For whatever reason, call it brain fatigue, I was returning `NIL` at the end of the `-provideNewView:` method unless it was the initial request. I couldn't figure out for the life of me why my color picker would not display properly most of the time. Once I realized I needed to always return the view from this method, my problem was solved. It just goes to show you that even simple bugs can be brutal to find.

Stragglers

Let's look at our stragglers. We have a few last methods to examine.

The `-doRandomColor:` method in [Listing 11.8](#) is called when the user clicks the ? button. The first thing that this method does is create a new `NSColor` object using the `SSRandomFloatBetween` inline static function to generate its individual color components. Remember `SSRandomFloatBetween` from [Chapter 10](#), "Screen Effects"? Remember our initialization of the random number seed in [Listing 11.2](#)? We then call the `-setColor:` method of the color panel returned by the `-colorPanel` method. Remember that the color panel is saved by our `NSColorPicker` super class in the `-initWithPickerMask: colorPanel:` method in [Listing 11.2](#). In turn, our own `-setColor:` method, in [Listing 11.4](#), is called.

Listing 11.8 ThePicker -doRandomColor: Action Method in ThePicker.m

```

static __inline__ float SSRandomFloatBetween(float a, float b)
{
    return a + (b - a) * ((float)random() / (float) LONG_MAX);
}

- (IBAction) doRandomColor: (id) sender
{
    NSColor *theColor;
    [self logText:@"doRandomColor\n"];
    theColor = [NSColor colorWithCalibratedRed: SSRandomFloatBetween(0.0, 1.0)
        green: SSRandomFloatBetween(0.0, 1.0)
        blue: SSRandomFloatBetween(0.0, 1.0)
        alpha: SSRandomFloatBetween(0.0, 1.0)];
    [[self colorPanel] setColor: theColor];
}

```

The `-logText:` method in [Listing 11.9](#) is a utility method used to display the name of the method being called in the Messages display. It merely accepts an `NSString` parameter and appends it to the `NSTextView` instance variable, `m_textView`.

Listing 11.9 ThePicker -logText: Method in ThePicker.m

```
- (void)logText: (NSString *)theString
{
    // Append the text to the end of the text view and scroll it into view
    NSRange theEnd = NSMakeRange([[m_textView string] length], 0);
    [m_textView replaceCharactersInRange: theEnd withString: theString];
    theEnd.location += [theString length];
    [m_textView scrollRangeToVisible: theEnd];
}
```

I told you that there wasn't much to it once you saw it all together.

[\[Team LiB \]](#)

◀ PREVIOUS

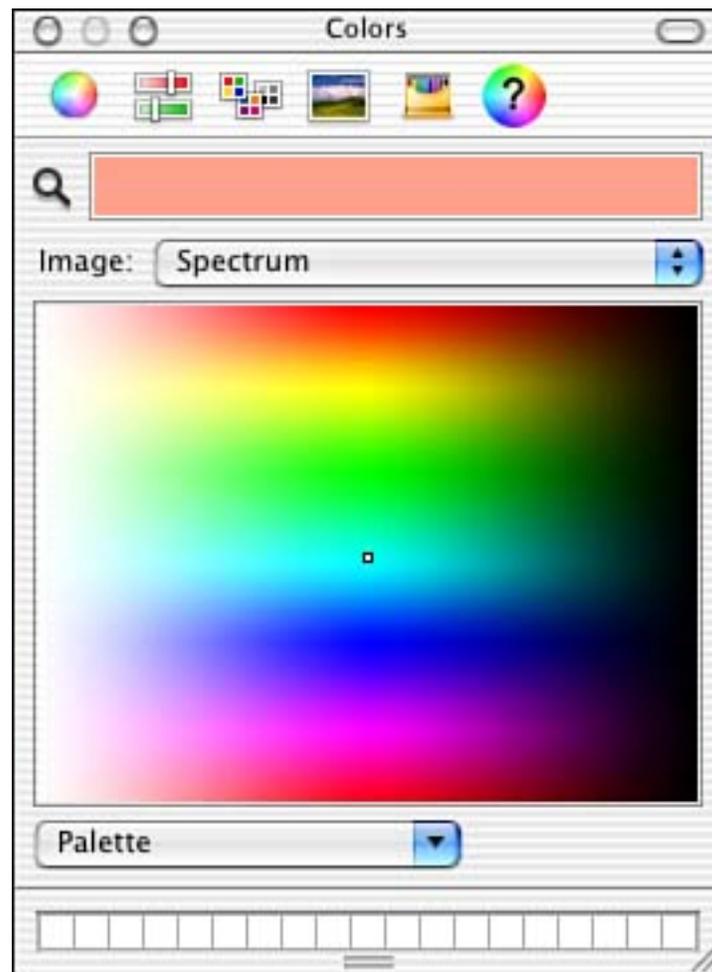
NEXT ▶

Try This

This is a simple color picker; it picks the color by choosing random values for the various color components. Let's see how we can spruce it up a bit.

Store a colorful image in the color picker and display it in the view. If the user clicks on the image, the pixel at the point that is clicked is sampled for the color and that color is made current. Your image can be anything you like. In fact, [Figure 11.8](#) shows how the Image Palettes color picker works.

Figure 11.8. The Image Palettes color picker.



Allow the user to choose a random color that weighs more toward red, green or blue. You can provide a pop-up button to allow the choice of which direction to weigh. You can then simply multiply the value of the chosen component (red, green, or blue) by a specific amount to make the color weigh in that direction. Maybe even allow the user to choose the weighting amount as well.

Conclusion

Well you've done great so far! Color Pickers are a fun type of code to write. You can provide all sorts of different schemes for choosing colors. Think about it and try to come up with the next great color picking mechanism. We have RGB, we have image palettes, and we've got color wheels; what's next? You decide!

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Part IV: Advanced Methods

[12 Threads](#)

[13 Terminal](#)

[14 XML-RPC](#)

[15 SOAP](#)

[16 Snippets](#)

[17 Conclusion](#)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 12. Threads

"Nice threads!"

—Some guy circa 1973

There was a time when computer programs ran from beginning to end with nary a surprise. If a program had 100 instructions, they would run in order, with the exception of loops and branches. No two instructions would execute at the same time. You could not be performing more than one task at the same time. Execution was serial; each command occurred one after the other.

The Macintosh operating system introduced us to the event driven programming model. A program's instructions would still occur in order, but this mechanism allowed the computer to spin its wheels until the user clicked the mouse or typed on the keyboard. When an event such as this occurred, the computer would react to it and draw text on the screen, make a window appear, and so on. However, the program managing all these events could still only do one thing at a time.

Enter threads. Conceptually, threads are flows of execution, which can span many functions or methods. Every process has at least one thread—every program you've seen in this book so far has used a single thread to run your code—but a process can spawn additional threads if necessary. Threads share the same address space as the process that spawned them; however, they are detached from the process and run independently. Most importantly, threads execute simultaneously, or asynchronously, and can share data with other threads in their process. Mind you, if your computer has only a single processor, only one instruction can actually execute at a time. Threads are most useful when multiple processors are available.

Note

Cocoa applications (and all processes) are made of threads. The main thread of an application is started when the application is launched. You can create other threads within the application using some simple methods of the `NSThread` class or the `pthread` API. You'll see how to work with threads using `NSThread` shortly.

Threads allow you to factor your application's functionality in order to provide a better user experience and performance, especially on multi-processor systems. By creating a thread of a time-consuming task, that task can be performed because time is available without blocking the rest of the program's execution. This is how the Finder copies files from disk to disk while you can simultaneously drag icons around and create folders.

Threads are also useful when you have a large task that would achieve better performance if it were divided into smaller tasks. This enables multi-processor systems to distribute the workload to achieve better efficiency. Because multiple threads can execute at the same time, more work can be accomplished in less time.

Threads do not come free, however. There is overhead associated with their use. On multi-processor systems, using threads for critical time-intensive tasks will usually yield favorable results. However, on a single-processor system, the results might be worse than if you hadn't used threads. You should carefully

benchmark your application using threads—then you should measure the quality of your application without them. You will see how much of a difference they actually make.

Note

If you are simply interested in deferring execution of a time-intensive task to make your application seem faster, you might consider using an **NSTimer** instead of an **NSThread**. I discussed **NSTimers** in [Chapter 3](#), "Cocoa Applications." Sometimes, deferring execution is all that is needed to make an application feel snappier to the user. Timers can even allow you to perform tasks in the background while the user continues doing other things in your application. However, if your background activity or user actions can't be broken up into quick operations, threads might be the answer.

[\[Team LIB \]](#)

◀ PREVIOUS

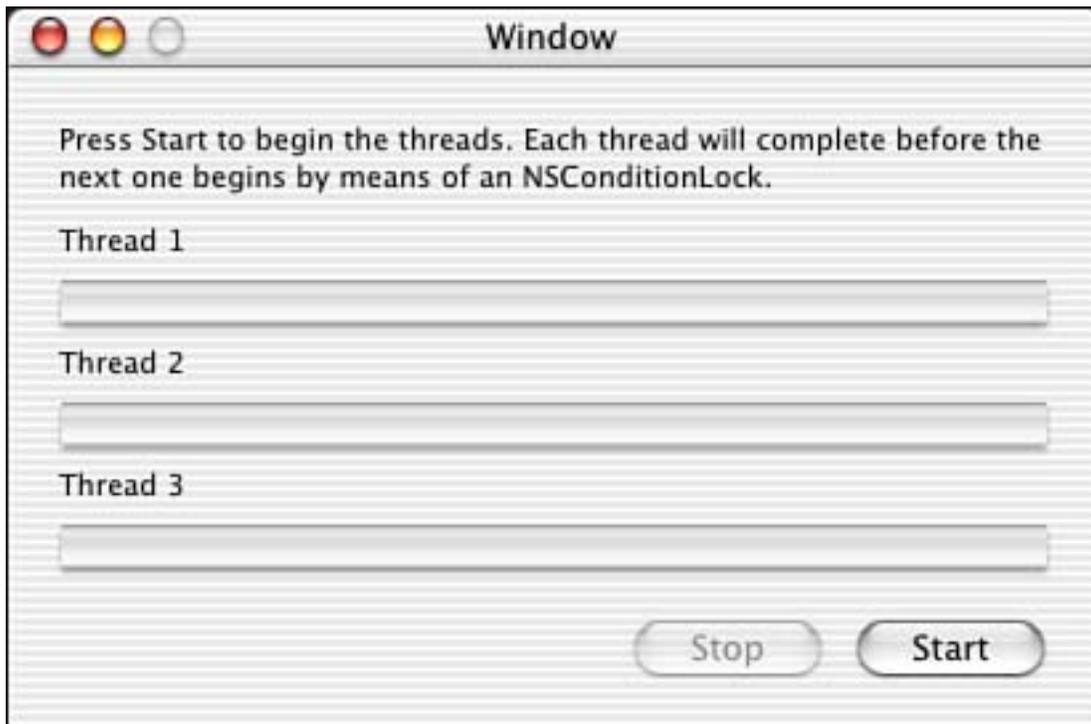
NEXT ▶

The Result

The project that we will be examining in this chapter shows you how to create and execute multiple threads. In the example, a thread will wait for the previous thread to complete before performing its job by use of an `NSConditionLock`. The threads will also access user interface elements during their execution.

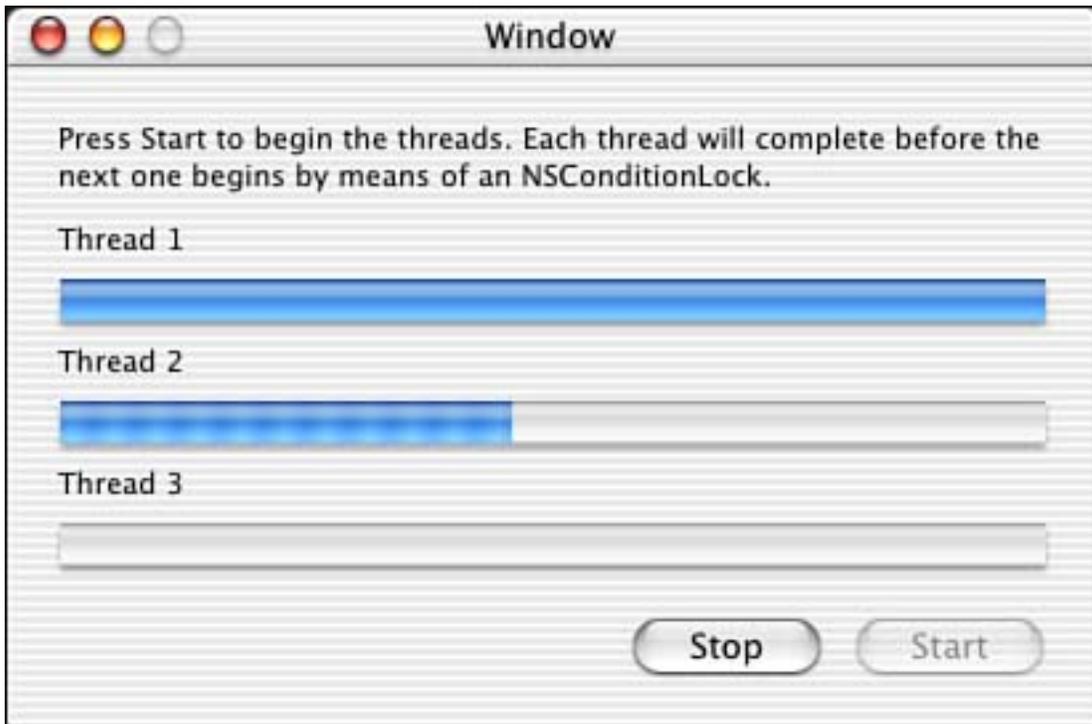
[Figure 12.1](#) shows the MyThread application before execution. Note the three progress indicators labeled as Thread 1 thru 3. The Start button is enabled, and the Stop button is disabled.

Figure 12.1. MyThread before execution.



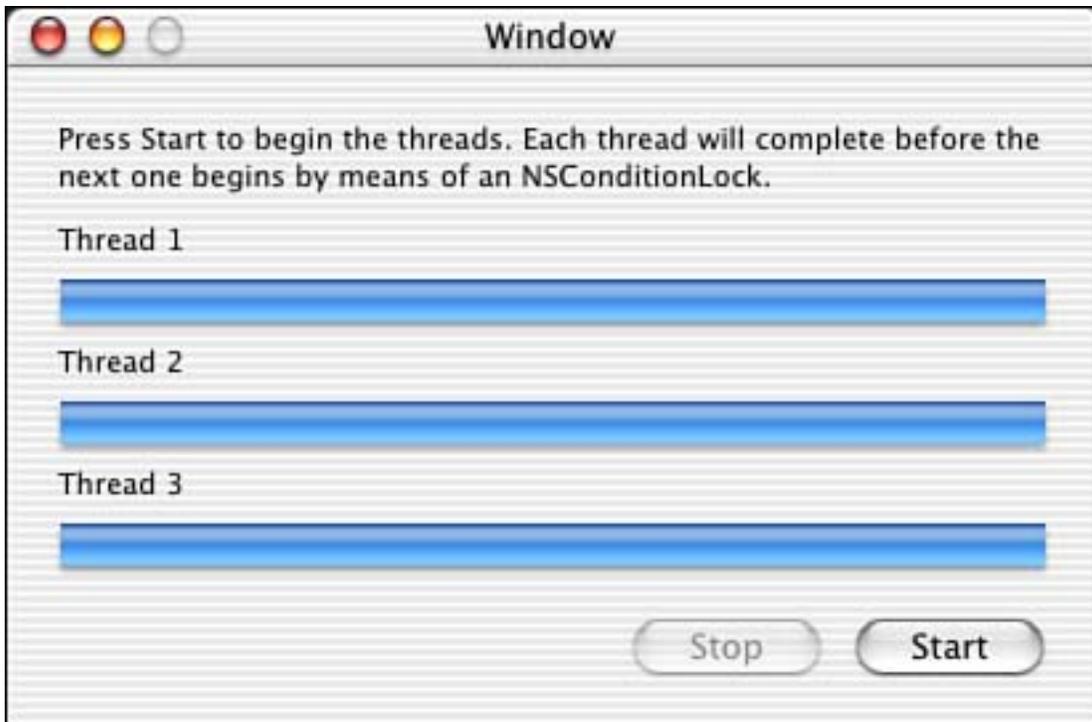
Upon pressing the Start button, the threads are created and executed in order, serially. That is, Thread 2 will not begin until Thread 1 has completed, and Thread 3 will not begin until Thread 2 has completed. The Start button is disabled, and the Stop button is enabled. This is shown in [Figure 12.2](#), where Thread 2 is currently executing.

Figure 12.2. MyThread during execution.



[Figure 12.3](#) shows the application once all threads have completed execution. The Start button is once again enabled to allow the process to begin again.

Figure 12.3. MyThread after execution.

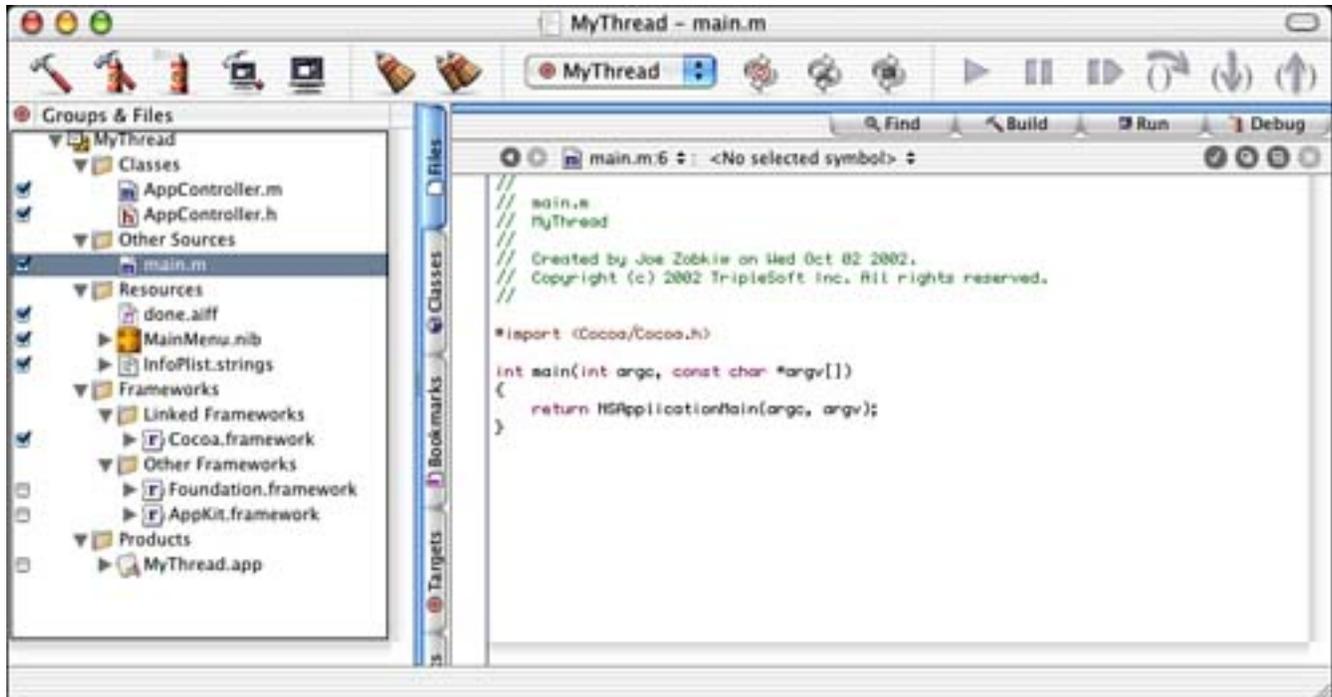


Let's look at the project.

The Project

[Figure 12.4](#) shows the MyThread project in Project Builder. The project is a simple Cocoa application. Note that we added an audio file to the project: `done.aiff`. You will see how to access that later; otherwise, there is nothing special about this project. All the code is in the `AppController` files that we will look at shortly.

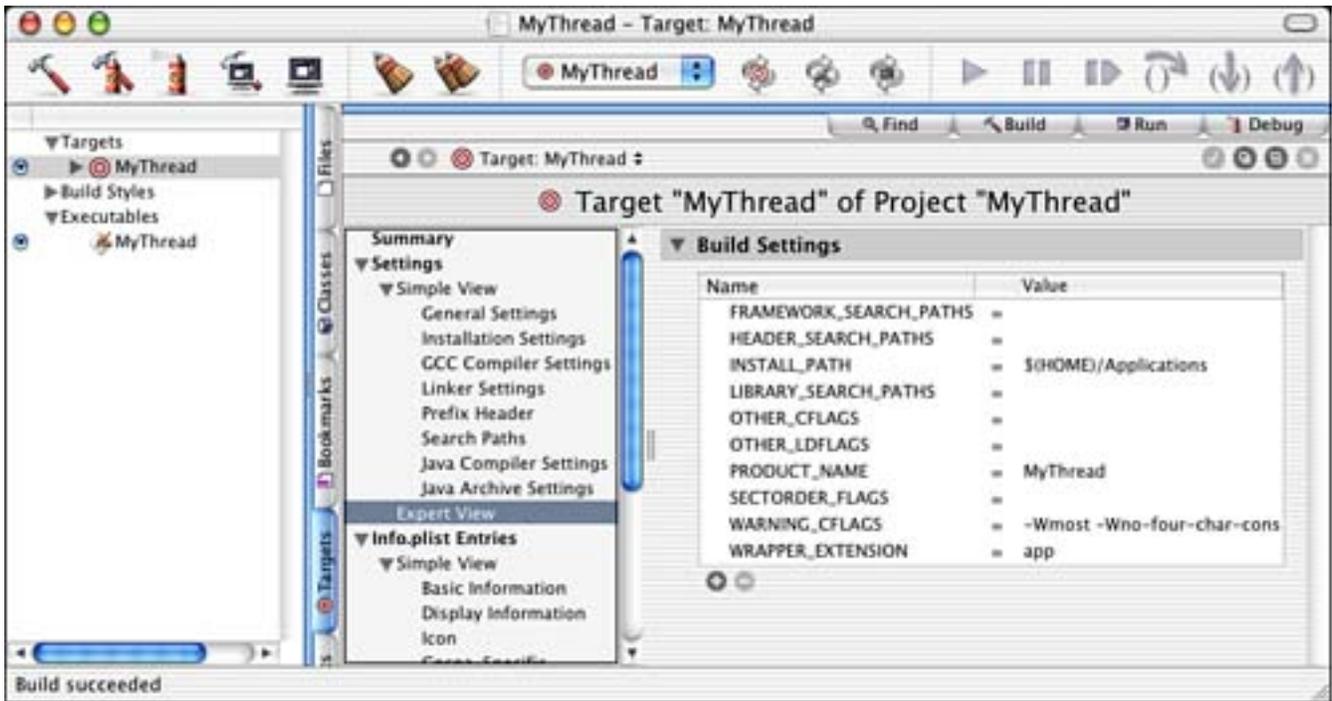
Figure 12.4. The MyThread project.



The Project Settings

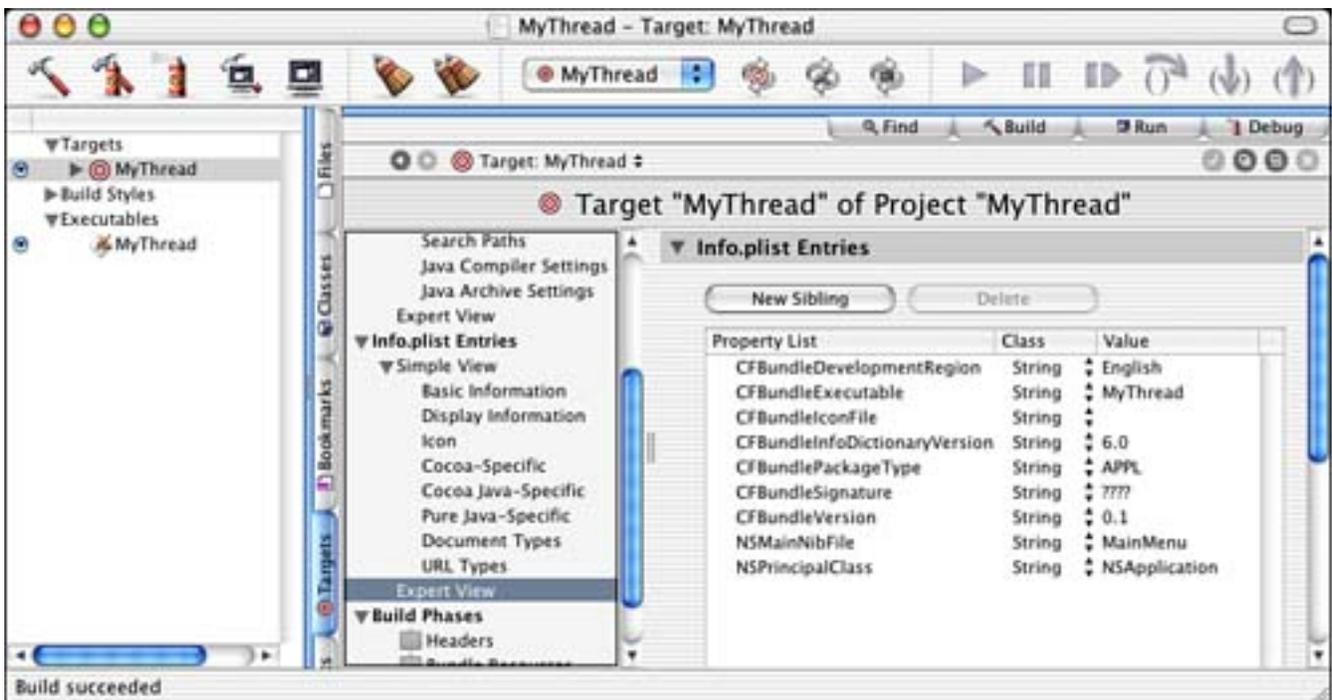
The Target settings in [Figure 12.5](#) are straightforward.

Figure 12.5. The MyThread Target settings.



The InfoPlist entries in [Figure 12.6](#) are also straightforward. Because this is a simple Cocoa application, there are no surprises here.

Figure 12.6. The MyThread InfoPlist entries.

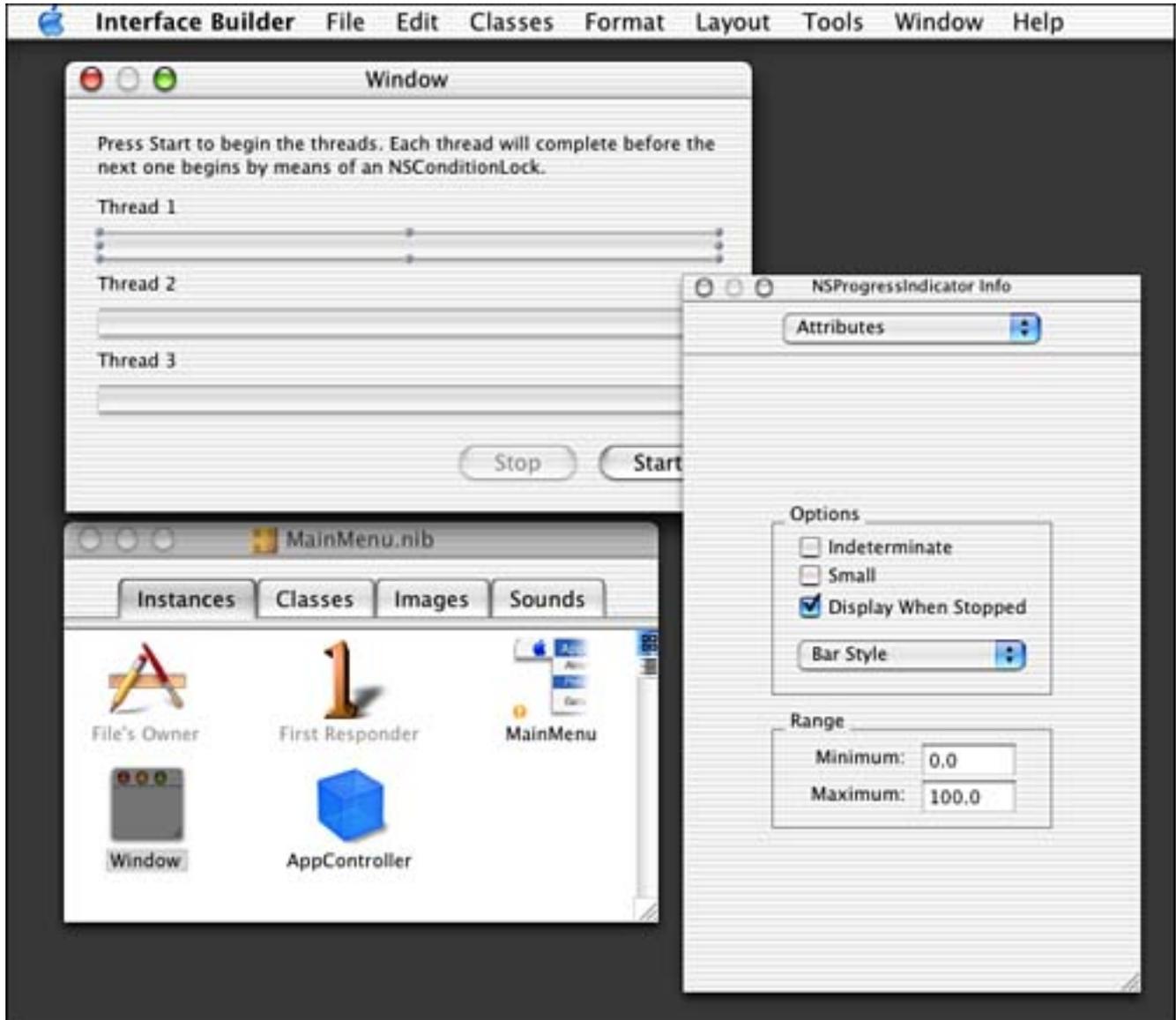


The Nib File

The `MainMenu.nib` file is shown in Interface Builder in [Figure 12.7](#). The `AppDelegate` has been instantiated and is the `NSApplication`'s delegate, as you would expect. We placed our controller methods for the window in the `AppDelegate` as well; although in a larger application, they would likely be in a

separate controller object. The main application window contains three `NSProgressIndicators` that we use to display the status and progress of our threads. You will see that the threads do nothing more than increment the value of their associated `NSProgressIndicator`—in the real world, you would do something more exciting. The Start and Stop buttons are connected to two `IBActions` named `-start:` and `-stop:`, respectively.

Figure 12.7. The MyThread MainMenu.nib file.



Let's go on to the source code!

The Source Code

The `AppController.m` and `AppController.h` files contain all the source code for your application. The interface of the `AppController` is shown in [Listing 12.1](#).

Listing 12.1 AppController Interface in AppController.h

```

#import <Cocoa/Cocoa.h>

@interface ApplicationController : NSObject
{
    IBOutlet NSButton *m_startButton;
    IBOutlet NSButton *m_stopButton;
    IBOutlet NSProgressIndicator *m_indicator1;
    IBOutlet NSProgressIndicator *m_indicator2;
    IBOutlet NSProgressIndicator *m_indicator3;

    NSConditionLock *m_conditionLock;
    BOOL            m_stop;
}
- (IBAction) start: (id) sender;
- (IBAction) stop: (id) sender;

- (void) thread1: (id) owner;
- (void) thread2: (id) owner;
- (void) thread3: (id) owner;

- (void) cleanupThread: (id) owner;

@end

```

Our `AppController` manages the `IBOutlet`s in the application's main window as well as the `IBActions` used to link to the Start and Stop buttons. We also define four methods that are used as thread entry points: `-thread1:`, `-thread2:`, `-thread3:`, and `-cleanupThread:`. You will see how these are created and executed next.

The `-start:` method in [Listing 12.2](#) is called when the user presses the Start button in the main window of the application. Assuming that we are not already in the middle of an operation, `m_conditionLock` is not `nil`. We begin the process of creating the threads and kicking them off, which means to begin their execution. The first thing we do is create and initialize an `NSConditionLock`. This class allows you to define conditions that your threads can then adhere to such as the states of a state machine. Each thread can then use the `NSConditionLock` to validate that the state machine is in the proper state before performing its job. Note that we initialize the `NSConditionLock` with a condition, or state, of `CONDITION_TASK_ONE`, defined later in [Listing 12.4](#).

Listing 12.2 `AppController -start:` Method in `AppController.m`

```

// This action is called when the Start button is pressed, it begins everything
- (IBAction) start: (id) sender
{
    if (m_conditionLock == nil) {
        // Start out in Condition 1 so things start up right away
        m_conditionLock = [[NSConditionLock alloc]
            initWithCondition: CONDITION_TASK_ONE];

        // Set stop flag
        m_stop = NO;

        // Create all the threads that will eventually run based on Condition
        [NSThread detachNewThreadSelector:@selector(thread1:)
            toTarget:self withObject:self];
    }
}

```

```

[NSThread detachNewThreadSelector:@selector(thread2:)
  toTarget:self withObject:self];
[NSThread detachNewThreadSelector:@selector(thread3:)
  toTarget:self withObject:self];

// Create a final, cleanup thread
[NSThread detachNewThreadSelector:@selector(cleanupThread:)
  toTarget:self withObject:self];

[m_startButton setEnabled:NO];
[m_stopButton setEnabled:YES];
} else {
  NSRunAlertPanel(@"Error", @"m_conditionLock is not nil",
    @"OK", @"", @"");
}
}
}

```

Note

A state machine is a way of keeping track of multiple conditions and acting on them accordingly. An application goes through several states when it runs. First, it initializes itself, it opens any documents that need opening, it loops waiting for user input, and eventually it quits and cleans itself up. Each of these can be considered a state.

In many programs, especially when following a specific communications protocol, it is important to know what state you are in at any time. For example, when transferring a file via FTP, you must open a connection to a remote host, attempt to log in, choose a file, begin the transfer, and so on. Although the actual FTP protocol is much more complex, you can see that keeping track of the state is important.

Next, we initialize our stop flag to NO. We use this flag to know whether the user pressed the Stop button. The threads, shown later, will check the value of this flag during their execution.

We then create our `NSThread` objects using the `+detachNewThreadSelector:toTarget:withObject:` class method of `NSThread`. This method creates and detaches a thread selector method, which exists in the specified target, passing a single argument to it. The thread selector method itself must be defined as follows: `-(void) aThread: (id) anArgument`. We create four threads in this manner—the three that track progress and then the last, `-cleanupThread:`.

Note

As the threads are created using `+detachNewThreadSelector:toTarget:withObject:` they are kicked off and immediately begin their execution. When the first thread of an application is created, the method posts the `NSNotification` with a `nil` object to the default notification center. This enables your application (or whatever object you desire) to know that it is using multiple threads.

Before exiting the method, we enable and disable the Start and Stop buttons appropriately. At this point, the job that the threads will perform has begun.

Before we look at the threads themselves, let's look at the `-stop:` action in [Listing 12.3](#). If the user presses the Stop button, we disable it and set the stop flag to `YES`. The threads, shown shortly, will check the value of this flag during their execution.

Listing 12.3 `AppController -stop:` Method in `AppController.m`

```
// This action is called when the Stop button is pressed,
// it sets a flag that the threads check
- (IBAction) stop: (id) sender
{
    [m_stopButton setEnabled:NO];
    m_stop = YES;
}
```

[Listing 12.4](#) defines our conditions. Each condition has an associated thread that executes during said condition. When the condition is `CONDITION_TASK_ONE`, `-thread1:` in [Listing 12.5](#) will execute. `-thread2:` in [Listing 12.6](#) will wait until the condition is `CONDITION_TASK_TWO` before it performs its job. This continues for all threads in the application. Let's look at a thread method in detail.

Listing 12.4 Condition Definitions in `AppController.m`

```
#define CONDITION_TASK_ONE        1
#define CONDITION_TASK_TWO        2
#define CONDITION_TASK_THREE      3
#define CONDITION_TASK_CLEANUP    4
```

Listing 12.5 `AppController -thread1:` Method in `AppController.m`

```
// This is the first thread to run, it starts as soon
// as the Start button is pressed
- (void) thread1: (id) owner
{
    int x;

    // Threads are responsible to manage their own autorelease pools
    NSAutoreleasePool *thePool = [[NSAutoreleasePool alloc] init];

    // Reset the progress indicator
    [m_indicator1 setDoubleValue:0];

    // Wait until Condition 1
    [m_conditionLock lockWhenCondition:CONDITION_TASK_ONE];

    // Loop, checking the stop flag
    for (x=1; x<=100; ++x) {
        if (m_stop) break;
        [m_indicator1 setDoubleValue:x];
        [NSThread sleepUntilDate:[NSDate
```

```

        dateWithTimeIntervalSinceNow: (NSTimeInterval) 0.05] ];
    }

    // Change to Condition 2
    [m_conditionLock unlockWithCondition: CONDITION_TASK_TWO];

    // Release the autoreleasepool
    [thePool release];

    // Exit this thread
    [NSThread exit];
}

```

[Listing 12.5](#) shows `-thread1:` which is the first thread to be executed in our application besides the application itself. All of our threads are modeled similarly in this example. The first thing to know about a thread is that it is responsible for its own `NSAutoreleasePool`. The main application thread normally handles this for your application, allowing you to allocate memory, mark it `autorelease`, and not worry about it. When you are in a custom thread, however, you are responsible for managing that memory on your own although you are sharing memory space with the application that created you. The thread is a guest in the application's home, but the application is not going to clean up after the thread. Therefore, we `alloc` and `init` a local `NSAutoreleasePool` to manage this memory issue.

Note

`NSApplication` contains a class method `+detachDrawingThread: toTarget: withObject:` that creates a new thread and automatically creates its `NSAutoreleasePool` for you. This is a convenience wrapper for `NSThread`'s `+detachNewThreadSelector: toTarget: withObject:`.

Next, we reset the progress indicator to zero and wait until the condition is `CONDITION_TASK_ONE`. Because the condition is initialized to `CONDITION_TASK_ONE`, as soon as this thread is called, the condition will match so that there will be no waiting to continue. By calling the `-lockWhenCondition:` method of `NSConditionLock`, we are taking control and saying, "We are using this lock now because our condition has been met—everyone else stay away!" You will see how the subsequent threads handle this situation shortly.

Note

You can also use `NSLock` instead of `NSConditionLock` if you are not concerned with multiple conditions as shown in this example. If you have only two conditions, an `NSLock` might suffice—it's locked, or it isn't.

We then loop 100 times to increment the value of our progress indicator user interface element. Note that we check the value of the `m_stop` instance variable. Remember, this is set to `YES` if the user presses the Stop button. In this case, we simply exit the loop and let the thread clean itself up. All subsequent threads exhibit this same behavior. Note also that we are using the `NSThread -sleepUntilDate:` method to pause the thread for a moment. If we didn't do this, the process would occur so quickly that it would make for a boring demo. Threads performing animation or other periodic tasks traditionally put themselves to sleep in order to conserve CPU time, battery life, and other sundries.

Note

Can you guess what thread the progress bar update is happening in? This background thread? No. When you increment the progress bar value, the control changes its internal state and marks itself as needing to be redisplayed. Later on, when the application is processing events on the main thread, the application will notice that the progress bar needs to be redisplayed, and it does so. In AppKit, most redisplay activity happens on the main thread. See the [NSView](#) documentation for more details.

After our loop is complete, our thread must notify the next thread awaiting execution. We do this by calling the `-unlockWithCondition:` method of `NSConditionLock` to not only unlock the lock, but also set the condition to the next state. This is saying, "We are done with this lock now and are setting it up for the next thread in line." Finally, we release our `NSAutoreleasePool` and `+exit` the thread to terminate ourselves.

Note

Calling `NSThread`'s `+exit` class method terminates the current thread. This is optional at the end of a thread, but could be called if a thread needed to be aborted prematurely. In either case, before exiting the thread, the `NSThreadWillExitNotification` notification is posted. The thread being exited is passed to the default notification center. All observers will receive this notification before the thread actually exits.

Now let's look at the next thread to execute. [Listing 12.6](#) shows `-thread2:`. You will note that this method is the same as `-thread1:`—with the exception that the condition it locks with is `CONDITION_TASK_TWO` instead of `CONDITION_TASK_ONE`, and the condition it unlocks with is `CONDITION_TASK_THREE` instead of `CONDITION_TASK_TWO`. It also updates the second progress indicator instead of the first—makes sense!

Listing 12.6 AppController `-thread2:` Method in AppController.m

```
// This is the second thread to run, it starts as soon as the Start button is
// pressed,
// then waits until the first thread is finished
- (void)thread2: (id)owner
{
    int x;

    // Threads are responsible to manage their own autorelease pools
    NSAutoreleasePool *thePool = [[NSAutoreleasePool alloc] init];

    // Reset the progress indicator
    [m_indicator2 setDoubleValue: 0];

    // Wait until Condition 2
    [m_conditionLock lockWhenCondition: CONDITION_TASK_TWO];

    // Loop, checking the stop flag
```

```

for (x=1; x<=100; ++x) {
    if (m_stop) break;
    [m_indicator2 setDoubleValue: x];
    [NSThread sleepUntilDate: [NSDate
        dateWithTimeIntervalSinceNow: (NSTimeInterval) 0.05]];
}

// Change to Condition 3
[m_conditionLock unlockWithCondition: CONDITION_TASK_THREE];

// Release the autoreleasepool
[thePool release];

// Exit this thread
[NSThread exit];
}

```

One thing to also note is that when this thread is initially created and detached, it actually executes immediately up until the `-lockWhenCondition:` method call. It then is blocked at that point until the lock is available with the chosen condition. This is the key to our state machine. Threads will get set up and ready, but will then wait for the proper condition before they execute. When finished, they notify the next thread by setting the next condition.

Note

Although our implementation works well, there is a way to optimize things here. We chose to create our threads all at once, which uses kernel resources to hold the data structures for five threads simultaneously (main, thread1, thread2, thread3, and cleanup). Imagine if we were to spawn the next thread as each child thread terminated; you would only need data structures for three threads (main, the current thread, and the new thread). Taking it one step further, if you were to spawn the next thread from the main thread after the background thread terminated, you would only need data structures for two threads (main and the new thread). See the section on wired memory costs in "Inside Mac OS X: Performance" at <http://www.apple.com/developer/> for more information on optimizations such as this.

Note that as soon as a thread calls `-unlockWithCondition:`, it is possible that the next thread picks it up and starts its work before the current thread even releases its `NSAutoreleasePool`.

Note

Imagine two threads attempting to read and write the same data without the benefit of locks. `NSLock` and `NSConditionLock` enable threads to work together, as a team, staying out of each other's way. Without them, threads accessing the same data source might step all over one another, attempting to read data that the other one hasn't even finished writing!

By using these classes, you are able to tell other threads, "Hey! I'm working on something here, so please let me finish unhindered. I will unlock when I'm through; then someone else can take over."

Also, look at the other lock options, including `NSRecursiveLock` and `NSDistributedLock`.

[Listing 12.7](#) is yet another thread, `-thread3:`. This thread is the same as `-thread2:`—with the exception of the conditions it locks and unlocks with and the progress indicator that it updates. Note that the unlock condition this time is `CONDITION_TASK_CLEANUP`. Let's see what the `-cleanupThread:` does.

Listing 12.7 `AppController -thread3:` Method in `AppController.m`

```
// This is the third thread to run, it starts as soon
// as the Start button is pressed,
// then waits until the second thread is finished
- (void)thread3: (id)owner
{
    int x;

    // Threads are responsible to manage their own autorelease pools
    NSAutoreleasePool *thePool = [[NSAutoreleasePool alloc] init];

    // Reset the progress indicator
    [m_indicator3 setDoubleValue: 0];

    // Wait until Condition 3
    [m_conditionLock lockWhenCondition: CONDITION_TASK_THREE];

    // Loop, checking the stop flag
    for (x=1; x<=100; ++x) {
        if (m_stop) break;
        [m_indicator3 setDoubleValue: x];
        [NSThread sleepUntilDate: [NSDate
            dateWithTimeIntervalSinceNow: (NSTimeInterval) 0.05]];
    }

    // Change to Condition 4
    [m_conditionLock unlockWithCondition: CONDITION_TASK_CLEANUP];

    // Release the autoreleasepool
    [thePool release];

    // Exit this thread
    [NSThread exit];
}
```

The `-cleanupThread:` method in [Listing 12.8](#) is the last thread to execute in our example. Once again, the thread is responsible for the `NSAutoreleasePool`. In addition, because it is the last thread, it can `-unlock` and `-release` the `NSConditionLock`. Note that it does not have to `-unlockWithCondition:` because there are no further conditions to set, so it can simply `-unlock`. We then set the `m_conditionLock` instance variable to `nil` so that the `-start:` method will have an easier time of things.

Listing 12.8 `AppController -cleanupThread:` Method in `AppController.m`

```

// This is the cleanup thread, it starts as soon as the Start button is pressed,
// then waits until the third thread is finished
- (void)cleanupThread: (id) owner
{
    // Threads are responsible for manage their own autorelease pools
    NSAutoreleasePool *thePool = [[NSAutoreleasePool alloc] init];

    // Wait until Condition 4
    [m_conditionLock lockWhenCondition: CONDITION_TASK_CLEANUP];

    // Last stop, clean up
    [m_conditionLock unlock];
    [m_conditionLock release];
    m_conditionLock = nil;

    // Update the UI
    [m_stopButton setEnabled: NO];
    [m_startButton setEnabled: YES];

    // Play done sound if the user didn't stop prematurely
    if (!m_stop)
        [[NSSound soundNamed:@"done"] play];

    // Release the autoreleasepool
    [thePool release];

    // Exit this thread
    [NSThread exit];
}

```

We then update the buttons properly so that Stop is disabled and Start is once again enabled. What is this `NSSound` stuff? As a bonus to the user, if he waited for the entire process to complete, a little sound is played. Remember the `AIFF` file we added to the project? The `NSSound +soundNamed:` class method and the `play` method allow us to quickly and easily play any sound available with the given name.

Finally, we release the `NSAutoreleasePool` and `exit` our thread.

One last notification method is `-windowWillClose:` in [Listing 12.9](#). This is called when our window is about to close (that is; the user clicked the Close box on the window). By calling our own stop method, this gives the application a chance to clean up its threads, just as if the user clicked the Stop button first and then closed the window!

Listing 12.9 `AppController -windowWillClose` Method in `AppController.m`

```

- (void)windowWillClose: (NSNotification *) aNotification
{
    [self stop:self];
}

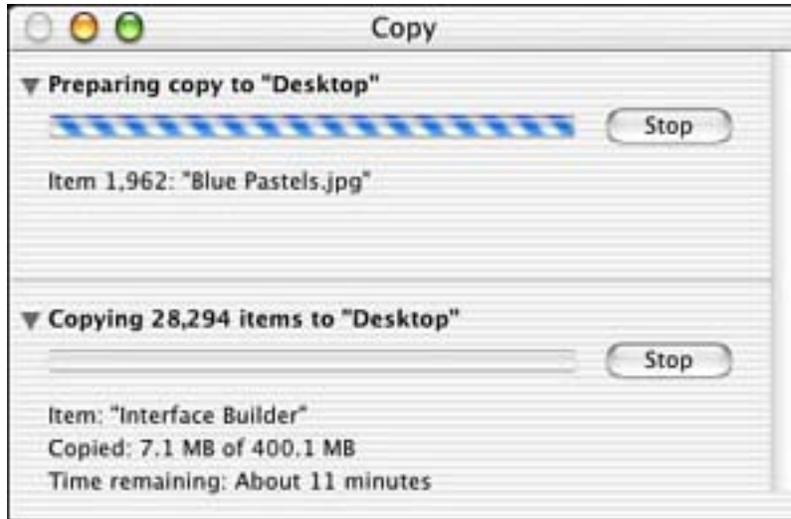
```

What do you think of my threads? Let's review a couple more aspects of threads before you pass judgment!

Serial Versus Parallel

The example in this chapter shows how to use threads in a serial fashion. That is, they execute in order, one after the other. However, threads are also very useful in situations in which you have multiple tasks that must occur simultaneously. Consider the example of the Finder with multiple copy jobs occurring at the same time. [Figure 12.8](#) shows the Finder in the midst of multiple copy jobs—each one is a thread!

Figure 12.8. The Finder making multiple copies.



[Figures 12.9](#) and [12.10](#) illustrate threads being used in both serial and parallel mechanisms.

Figure 12.9. Threads being used serially.

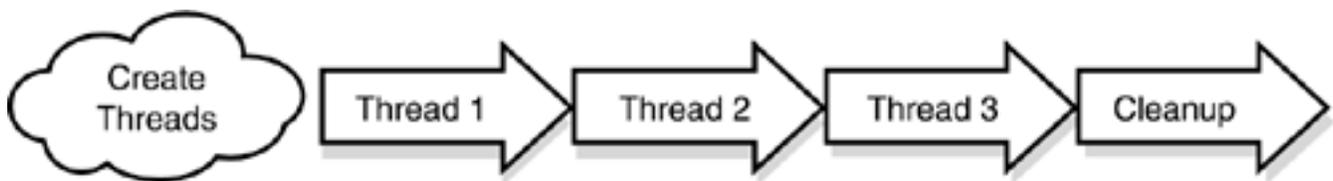
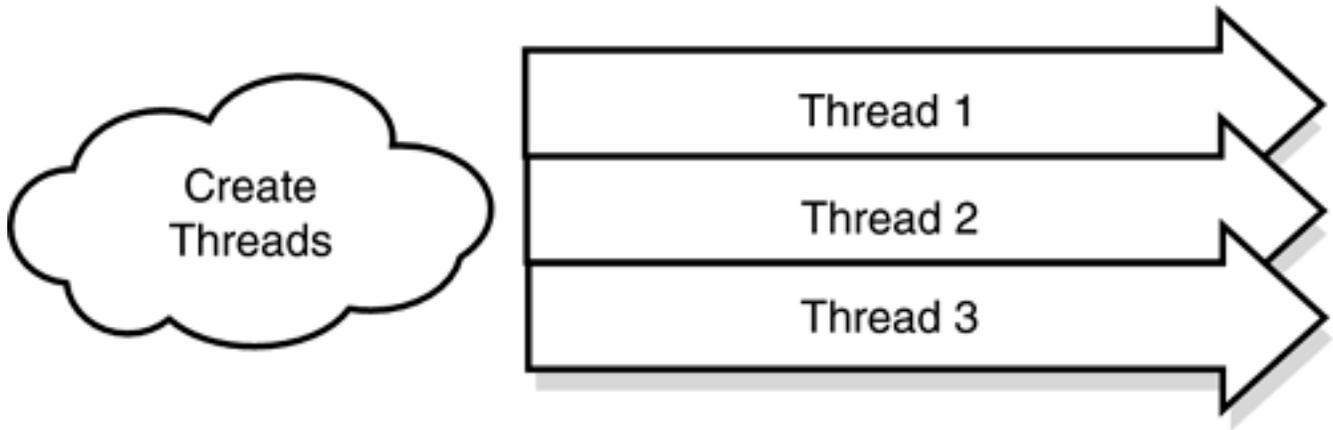


Figure 12.10. Threads being used in parallel.



In [Figure 12.9](#), the threads are all created and then executed one after the other, serially. Well, this is not completely true in our example. Remember, the threads are executed the moment they are created, prepare themselves, and then wait for the right condition before actually performing the bulk of their work. However, you get the general idea. The point of this illustration is that one thread does not perform its task until the thread before it has finished its task.

In [Figure 12.10](#), the threads are all created and then execute at the same time, in parallel. Note also that although the threads are all running simultaneously, at any given time one or more of them might be sleeping or blocking on IO and not actually doing any work. Remember, sleeping threads give other threads a chance to work. The point of this illustration is that all threads can simultaneously perform their tasks as needed.

[\[Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

Thread Priority

Although all the threads in this example were of the same priority, there are times when you might need to set the priority of a thread above that of another in your application.

For example, assume that you have three threads running in parallel that perform certain tasks. One thread might check for the existence of a file in a certain location and then put itself to sleep for 30 seconds using the `+sleepUntilDate:` class method of `NSThread`. Another might run constantly waiting for specific data input via a USB device, sleeping only for a few milliseconds at a time. The last might run constantly processing the data input, also sleeping for only a few milliseconds at a time.

Of these last two threads, the input processor might be the most important and be given the highest priority—you don't want to miss any input from a hardware device. By using the `+setThreadPriority:` class method of `NSThread`, you can set the priority of any thread in relation to other threads throughout the operating system. The range of values is 0.0 to 1.0 for the priority, where 1.0 is the highest priority. The operating system manages the scheduling of these threads based on overall load. The `+threadPriority` class method will return the priority of any particular `NSThread`.

Note

Note that using notifications can also curb the need for this type of polling. Notifications enable your thread to be notified when data becomes available or some other condition is true. Threads register themselves as observers of a notification and simply wait for the notification to be delivered. This is similar to waiting for the phone to ring: You don't pick up the phone every minute to see whether someone is there for you; you wait for it to ring, and then you pick it up! See the `NSNotification` documentation for more information. Also see [Chapter 13](#), "Terminal," for more hints on using notifications.

Thread Data

The `-threadDictionary` instance method of `NSThread` returns the threads dictionary, an `NSMutableDictionary`. This dictionary can be used to store any data specific to the thread. Because the dictionary is available to anyone who has access to the `NSThread` instance itself, this can be a very useful way to pass information to and from a thread. One caveat is that accessing a thread's dictionary might be slightly slower than accessing an `NSMutableDictionary` that is not associated with a thread.

Try This

Here are some things to try to expand the project in this chapter:

Instead of creating all threads in the `-start:` method and using condition locks, try this: Create the first thread in the `-start:` method and have it create the second thread when it has completed execution. Then have the second thread create the third when it is complete, and so on. Using this mechanism, you can avoid the use of the `NSConditionLock`.

Add an indeterminate progress indicator to the user interface and create a thread that runs in parallel with the other threads running serially. This new thread should constantly call the `NSProgressIndicator's -animate:` method while the others are performing their tasks.

Conclusion

Congratulations for making it through this chapter. You can see that this technique isn't too complex, but it might take a bit of time to wrap your head around it. Using simple techniques such as this can really improve the structure and implementation of your application. Using condition locks to drive your state machine is a great way to manage a program's work load and the individual steps of a process—especially when they so closely depend on one another.

There is a world of threads beyond what is covered in this chapter. You should explore the documentation supplied by Apple for other techniques that can be used with them. Using threads is a very powerful technique that can make your application performance soar when used correctly.

Chapter 13. Terminal

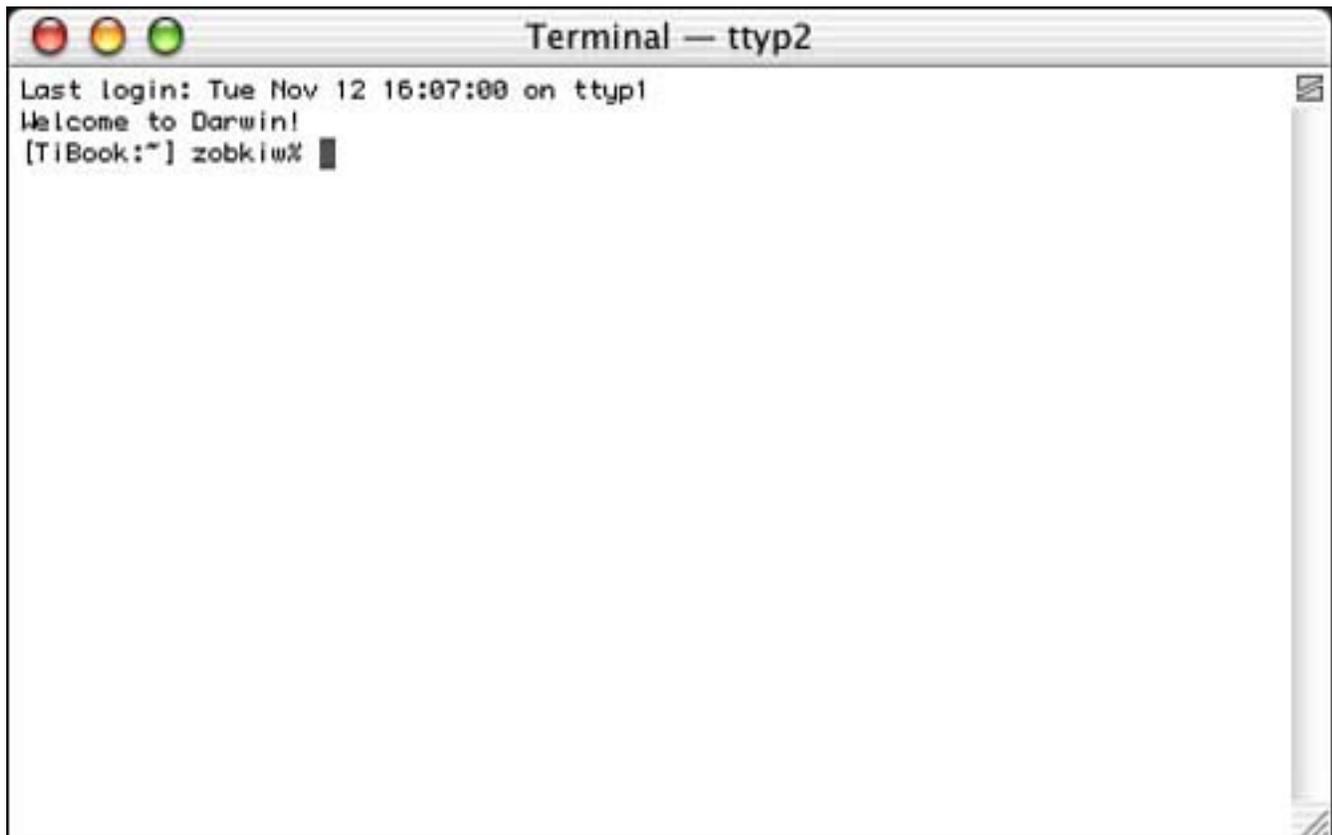
"No manual entry for help."

—man help

A terminal is nothing more than a connection to a computer that allows you to type commands and have results displayed back to you. Terminals existed before the graphical user interface took hold. UNIX was originally a terminal-based operating system. DOS still is. Mac OS X offers the power of the terminal along with its elegant Aqua user interface allowing you to achieve greater control over your computer than you can with a graphical user interface alone.

[Figure 13.1](#) shows the Terminal application packaged with Mac OS X. This application gives you all the options you would expect from a standard UNIX terminal. Using the terminal, you can perform file manipulation, process control, remote computer control, execute shell scripts, and more. The terminal is your window to a world in which there is no menu bar and no cute icons. Enter if you dare, but behold and respect the power it possesses.

Figure 13.1. Terminal application in Mac OS X at login.



Note

Mac OS X is still a terminal-based operating system. Do this: In the System Preferences'

Accounts' Login Options tab, switch to "Name and Password" and turn off auto-login in the Users tab. Now log out. Log in as `>console` with no password; note the use of the greater than sign here. You're looking at Mac OS X's console. Simply type `Logout` to exit the console.

Let's look at some of the things you can do in the terminal.

You can easily copy files using wildcards. That is, instead of picking and choosing files one by one in the Finder, you can choose files by name or partial name. To find all files that end in `.TIFF`, you would represent that as `*.TIFF` in your command.

Note

In fact, you can even type multiline commands in the terminal. For example, the following command renames all files in the current directory that end in `.TIFF` to end in `.TIF` using the `sed` program.

```
foreach file ( *.TIFF )
mv $file `echo $file | sed 's/.TIFF/.TIF/'`
end
```

The `sed` program is a stream editor that modifies its input and writes it back out. This is a very powerful program. `Grep` is another powerful program that can be used on the command line as well. Type `man sed` or `man grep` in Terminal for more information about these programs.

You can log in to a remote computer and, if you have the proper permissions, perform any task you can on your local computer. The secure shell (`ssh`) is a popular mechanism to achieve an encrypted connection from one computer to another. You would simply type `ssh some.computer.com` to connect. You can then use secure copy (`scp`) to securely copy files between the two machines. This is a great (secure) alternative to FTP.

You can also execute shell scripts to handle everything from backups to cache file cleanup or from automatic Web site updating from your local computer to your Web server. These scripts can be executed manually or on a schedule via the `cron` application. Running `top`, as shown in [Figure 13.2](#), you can track processor usage by application.

Figure 13.2. Terminal application in Mac OS X running `top`.

```
Terminal — ttyp2
Processes: 61 total, 2 running, 59 sleeping... 172 threads      16:09:41
Load Avg: 0.61, 0.73, 0.62   CPU usage: 15.1% user, 15.1% sys, 69.7% idl
SharedLibs: num = 7, resident = 2.21M code, 244K data, 560K LinkEdit
MemRegions: num = 6508, resident = 204M + 13.5M private, 164M shared
PhysMem: 67.8M wired, 125M active, 430M inactive, 623M used, 145M free
VM: 2.80G + 3.62M 10857(0) pageins, 5(0) pageouts

  PID COMMAND      %CPU   TIME    #TH  #PRTS  #MREGS  RPRVT  RSHRD  RSIZE  VSIZE
  514 top           12.6%  0:01.14  1    14     18     268K   348K   564K  13.6M
  492 tcsh           0.0%   0:00.05  1    10     15     344K   600K   788K   5.73M
  491 login          0.0%   0:00.66  1    12     33     244K   400K   572K  13.7M
  490 tcsh           0.0%   0:00.03  1    10     15     348K   600K   792K   5.73M
  489 login          0.0%   0:00.57  1    12     33     244K   400K   568K  13.7M
  485 BBEEdit 6.5    0.0%   0:05.47  3    74    140   3.87M+ 15.9M  9.72M+ 72.5M
  478 Terminal       0.0%   0:09.66  4    62    138   1.59M  12.9M  9.50M  63.2M
  453 csmount        0.0%   0:02.04  2    20     23     280K   688K   728K  14.5M
  444 Interarchy     1.6%   1:00.00  7   132    159   3.44M  18.3M  7.66M  71.9M
  434 SpeechServ     0.0%   0:00.85  4   103     99   2.97M  3.04M  4.35M  52.3M
  431 csmount        0.0%   0:00.22  2    19     23     240K   688K   680K  14.5M
  423 AppleSpell     0.0%   0:00.09  1    16     21     440K   856K   1.02M  14.6M
  421 Preview        0.0%   0:13.42  2    87    124   2.99M  15.6M  6.69M  68.1M
  420 Adobe Phot     8.4%  27:32.54  7    85    683   31.2M  36.8M  49.7M  146M
  416 Cocoa Brow    0.0%   0:13.17  1    60     91   2.28M  8.40M  5.36M  57.2M
```

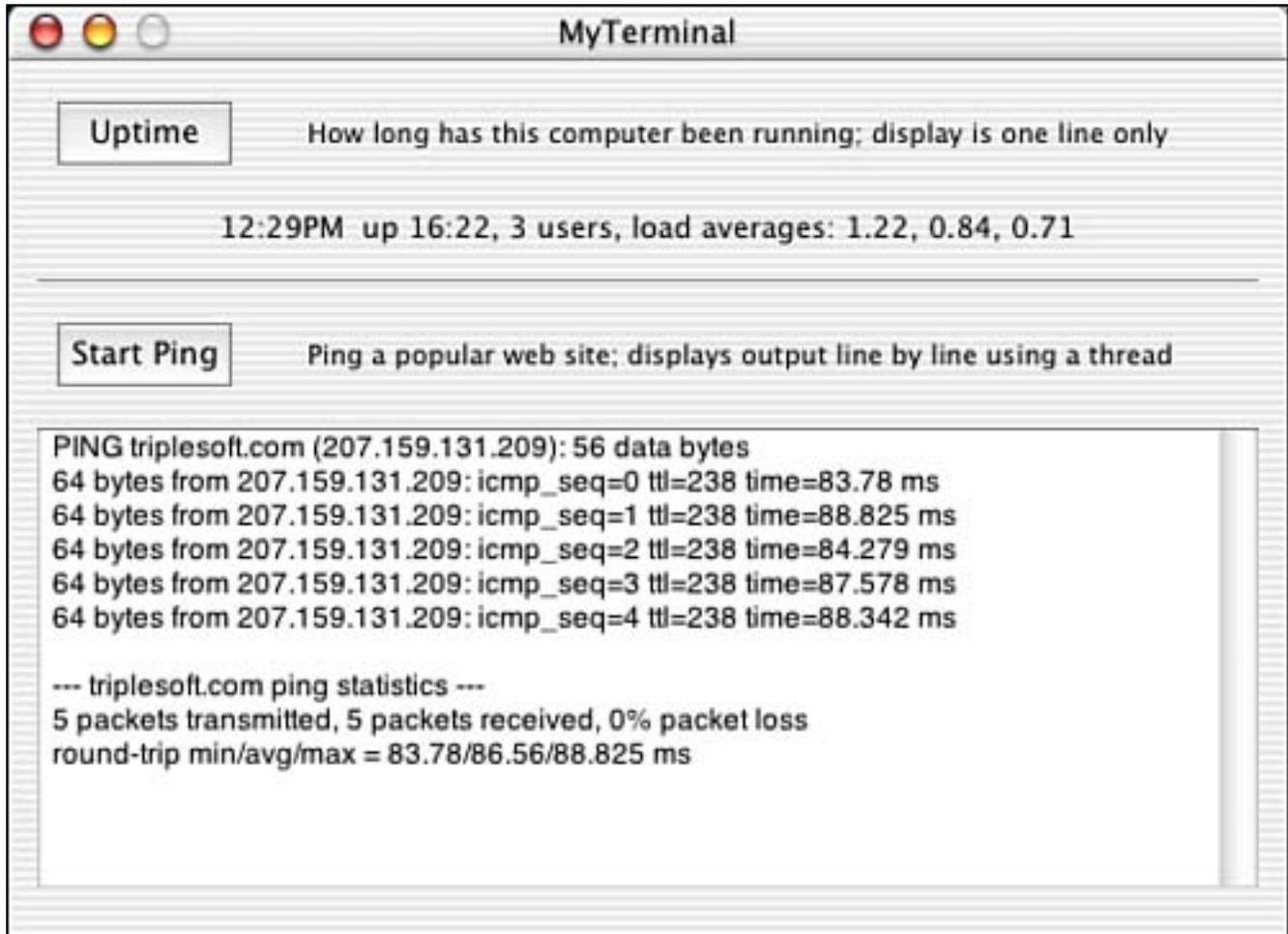
UNIX commands tend to be cryptic in nature and aren't written with novice users in mind. For example, the `rm` command removes files without warning or comment, and it's easy to delete more than you intended. In general, be wary of running commands without knowing what they do. You can learn what commands do using the `man` command. `man` is short for "manual," and it contains a lot of documentation on the command line tools and POSIX C libraries available on Mac OS X.

As you can see, the terminal is a powerful tool. There are times, however, when you might want to leverage this power from within your Cocoa application. This chapter looks at an application that does exactly that.

The Result

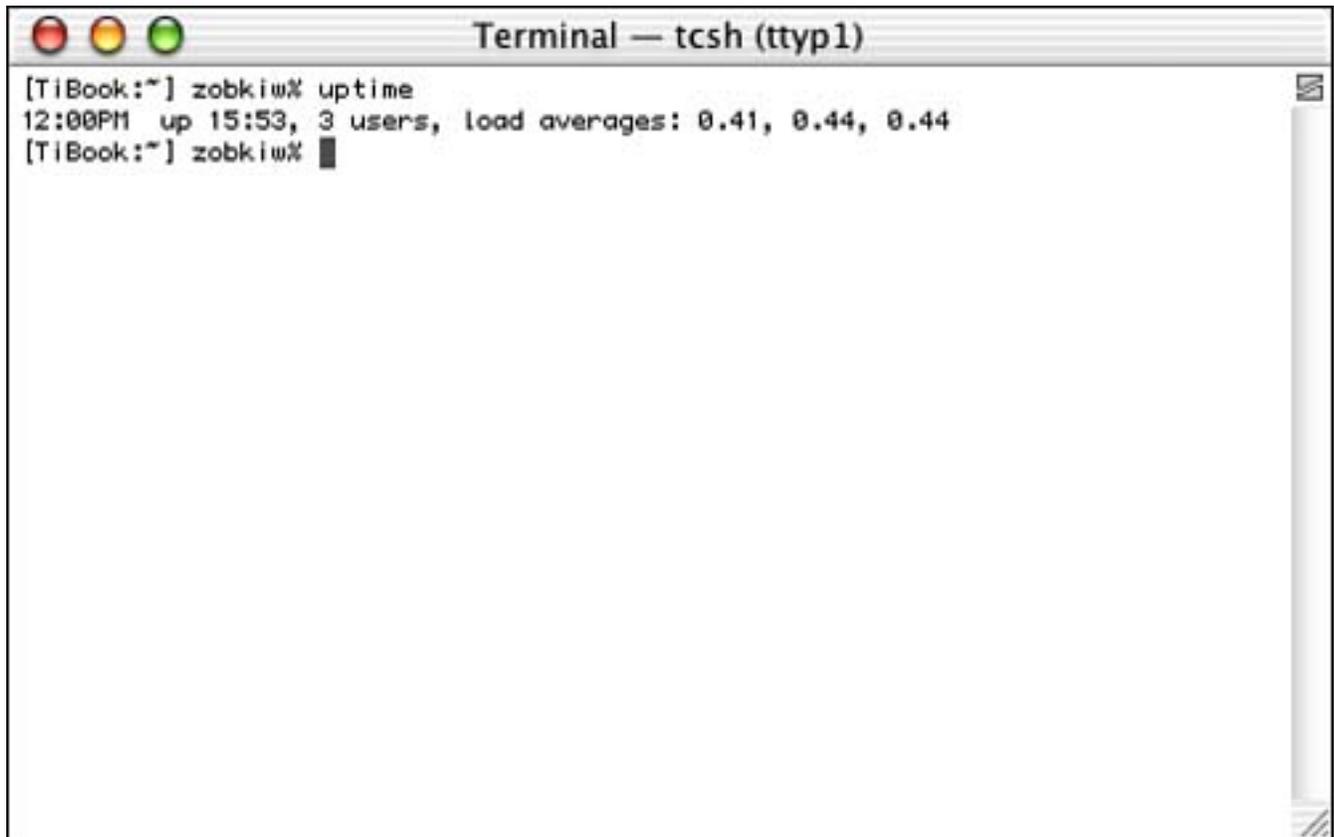
MyTerminal, shown in [Figure 13.3](#), accesses the power of the terminal in a few different ways using the `NSTask` and `NSPipe` classes.

Figure 13.3. The MyTerminal application.



MyTerminal first interfaces with the `uptime` program. Uptime is a simple command-line application that takes no parameters and reports the current time, the length of time the system has been up, the number of users, and the load average of the system over the last 1, 5, and 15 minutes. In the example in [Figure 13.3](#), it is currently 12:29 p.m., the system has been up for 16 hours and 22 minutes, 3 users are on the system, and the load average has increased from 0.71% 15 minutes ago to 1.22% 1 minute ago. Pressing the Uptime button calls the program and reports the results. [Figure 13.4](#) shows what uptime looks like running in the terminal.

Figure 13.4. Uptime running the terminal.

A screenshot of a terminal window titled "Terminal — tcsh (tty1)". The window has three colored window control buttons (red, yellow, green) in the top-left corner. The terminal content shows the user [TiBook:~] entering the command 'uptime'. The output is '12:00PM up 15:53, 3 users, load averages: 0.41, 0.44, 0.44'. The prompt returns to [TiBook:~] zobkiw% with a cursor. A vertical scrollbar is visible on the right side of the terminal window.

```
[TiBook:~] zobkiw% uptime
12:00PM up 15:53, 3 users, load averages: 0.41, 0.44, 0.44
[TiBook:~] zobkiw% █
```

Second, MyTerminal interfaces with the `ping` program, as shown in [Figure 13.3](#). Ping is a more complex command-line application that takes multiple parameters—most of them optional—sends a packet to a remote computer, and reports the response. This program is used in network debugging to tell if another computer is reachable. Ping sends an `ECHO_REQUEST` packet to the remote computer; if it receives it, the `ECHO_RESPONSE` packet will be returned. Ping then displays the time it took to receive the response. This can be a key indicator to the health of a network or portion thereof. Pressing the Start Ping button calls the program and reports the results. [Figure 13.5](#) shows what ping looks like running in the terminal.

Figure 13.5. Ping running the terminal.

```
Terminal — tcsh (tty1)
[TiBook:~] zobkiw% ping -c 10 triplesoft.com
PING triplesoft.com (207.159.131.209): 56 data bytes
64 bytes from 207.159.131.209: icmp_seq=0 ttl=238 time=84.928 ms
64 bytes from 207.159.131.209: icmp_seq=1 ttl=238 time=85.476 ms
64 bytes from 207.159.131.209: icmp_seq=2 ttl=238 time=84.818 ms
64 bytes from 207.159.131.209: icmp_seq=3 ttl=238 time=90.364 ms
64 bytes from 207.159.131.209: icmp_seq=4 ttl=238 time=88.344 ms
64 bytes from 207.159.131.209: icmp_seq=5 ttl=238 time=90.057 ms
64 bytes from 207.159.131.209: icmp_seq=6 ttl=238 time=89.617 ms
64 bytes from 207.159.131.209: icmp_seq=7 ttl=238 time=92.739 ms
64 bytes from 207.159.131.209: icmp_seq=8 ttl=238 time=90.109 ms
64 bytes from 207.159.131.209: icmp_seq=9 ttl=238 time=98.3 ms

--- triplesoft.com ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 84.818/89.475/98.3 ms
[TiBook:~] zobkiw% █
```

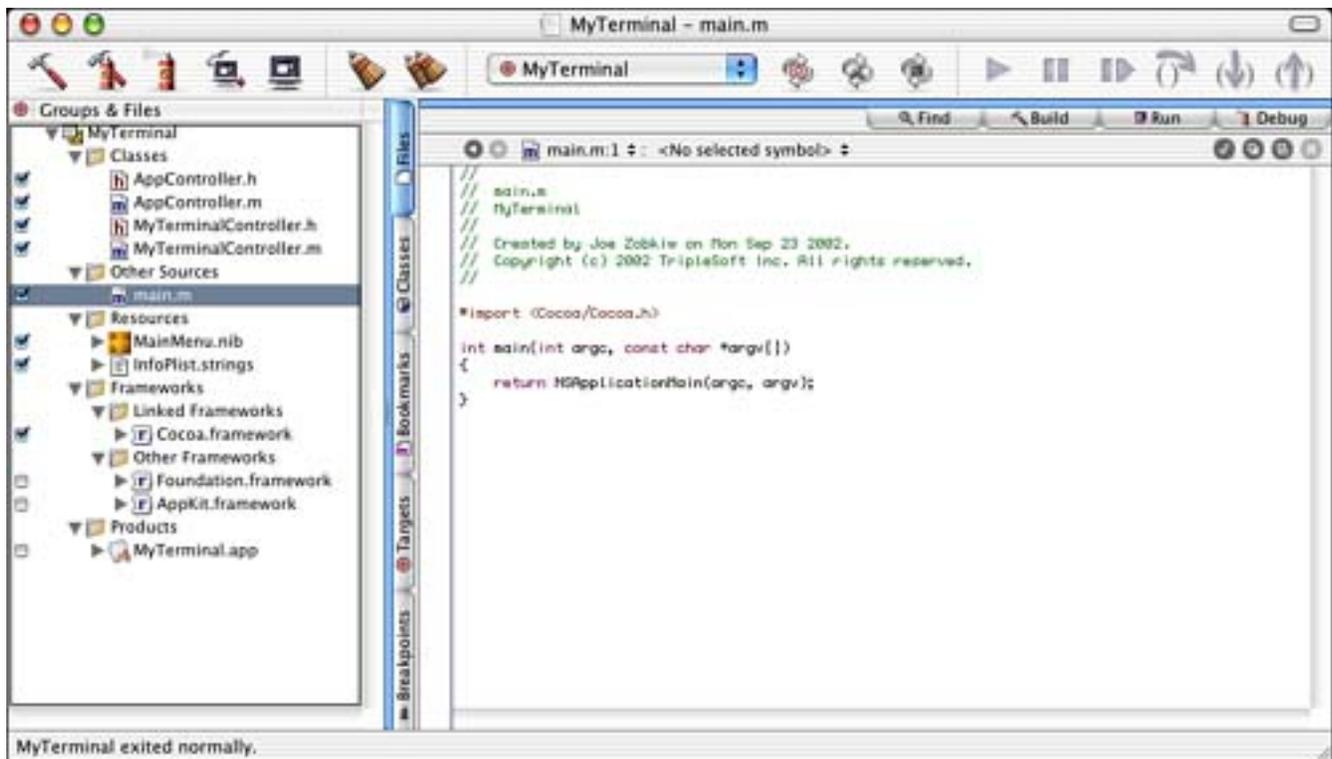
Let's look at the project to see what's going on here.

[\[Team LIB \]](#)

The Project

[Figure 13.6](#) shows the MyTerminal project in Project Builder. The project is a simple Cocoa application. We use an `AppController` simply to implement the `-applicationShouldTerminateAfterLastWindowClosed:` delegate method. We've done this in previous examples, so I won't spend time talking about it again here. `MyTerminalController` is an `NSObject` subclass that manages the user interface objects in our main application window.

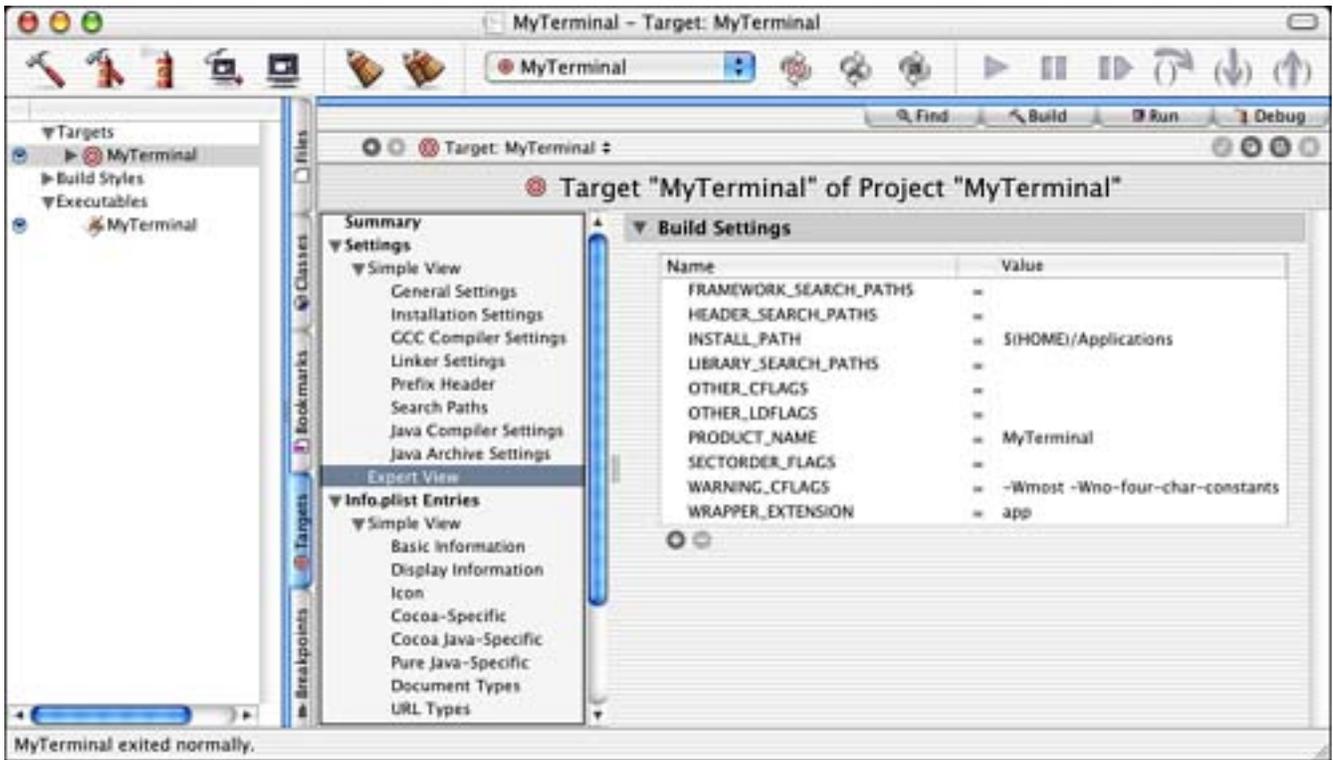
Figure 13.6. The MyTerminal project.



The Project Settings

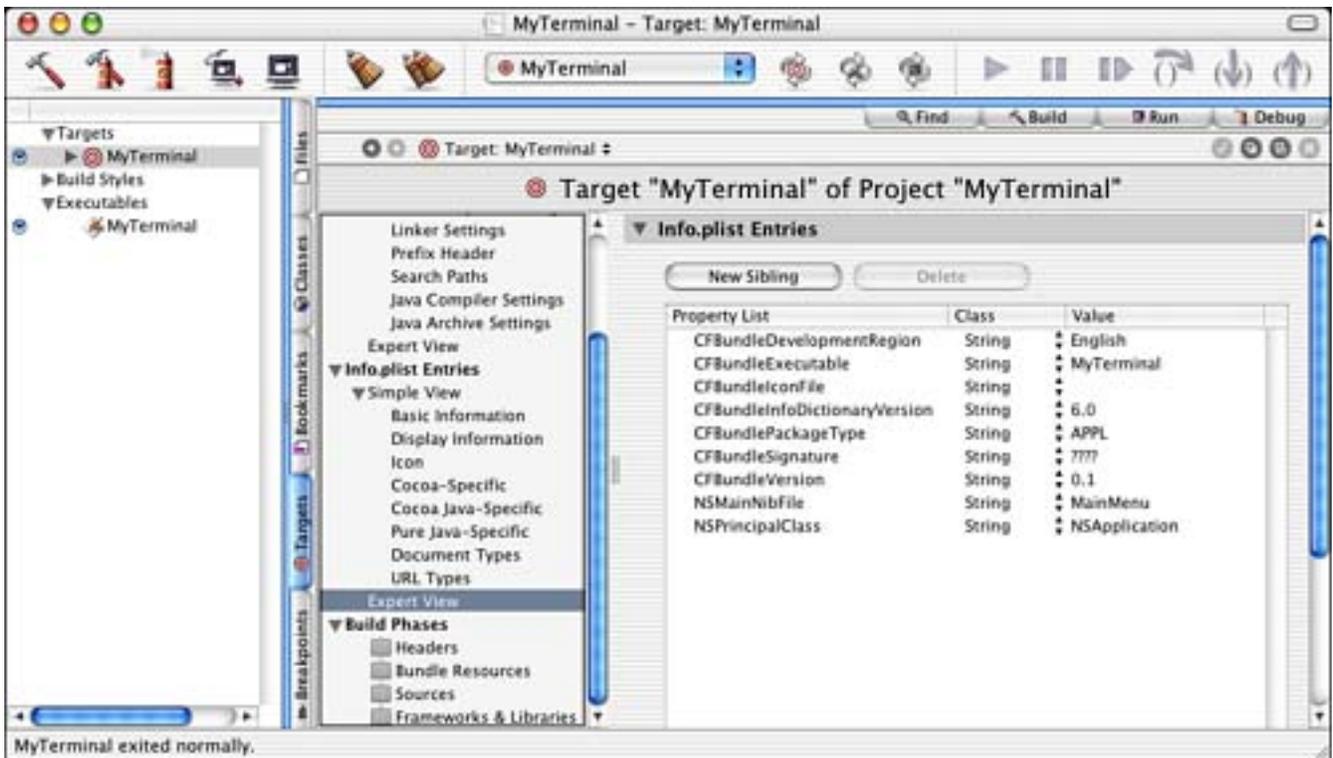
The Target settings in [Figure 13.7](#) are straightforward for a simple Cocoa application.

Figure 13.7. The MyTerminal Target settings.



The InfoPlist entries in [Figure 13.8](#) are also straightforward. Because this is a simple Cocoa application, there are no surprises here.

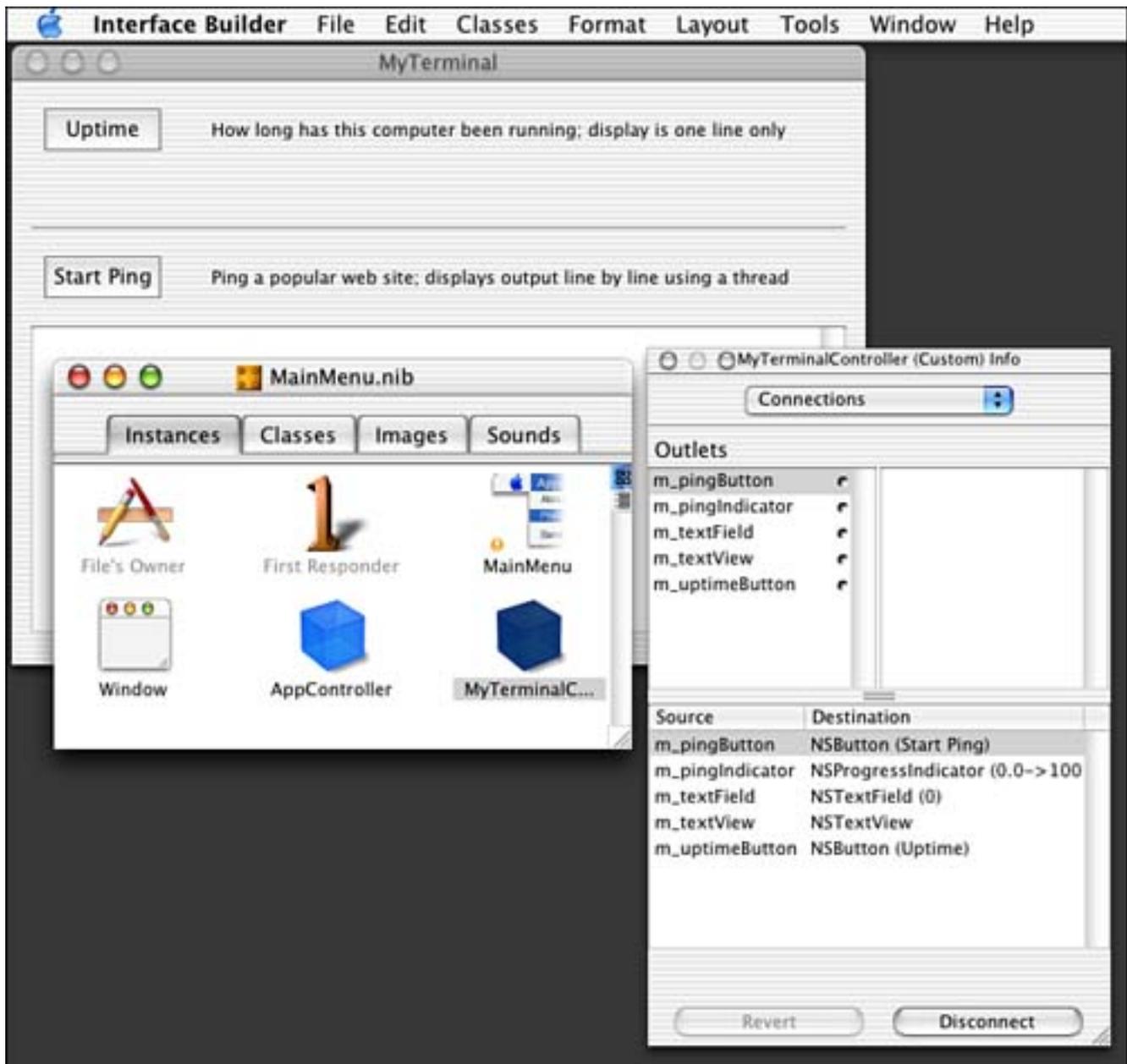
Figure 13.8. The MyTerminal InfoPlist entries.



The Nib File

The `MainMenu.nib` file is shown in Interface Builder in [Figure 13.9](#). The `AppController` has been instantiated and is the `NSApplication`'s delegate, as you would expect. The `IBOutlet`s and `IBAction`s that control the items in our main application window are in the `MyTerminalController` custom class. Items you cannot see in this picture are the `NSTextField` below the Uptime area of the window and the `NSProgressIndicator` to the right of the Start Ping button. The Uptime and Start Ping buttons are connected to two `IBAction`s named `-uptime:` and `-ping:`, respectively.

Figure 13.9. The `MyTerminal MainMenu.nib` file.



Let's look to the source code!

The Source Code

The interface of `MyTerminalController` is shown in [Listing 13.1](#).

Listing 13.1 MyTerminalController Interface in MyTerminalController.h

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

@interface MyTerminalController : NSObject
{
    IBOutlet NSTextView          *m_textView;
    IBOutlet NSButton           *m_pingButton;
    IBOutlet NSProgressIndicator *m_pingIndicator;

    IBOutlet NSTextField        *m_textField;
    IBOutlet NSButton           *m_uptimeButton;

    NSTask                      *m_pingTask;
    NSPipe                      *m_pingPipe;
    BOOL                        m_pingIsRunning;
}
- (IBAction) ping: (id) sender;
- (IBAction) uptime: (id) sender;

- (void) displayPingData: (NSFileHandle*) theFileHandle;

@end
```

The `MyTerminalController` manages the `IBOutlets` of our user interface as well as the `IBActions` of the two buttons mentioned previously. We also define a method called `-displayPingData:` that is actually detached and executed as a thread. If you have yet to read [Chapter 12](#), "Threads," you might consider doing so before continuing.

Note the `NSTask` and `NSPipe` instance variables. `NSTask` enables our program to run another program as a sub process and monitor its execution. `NSTask` creates a separate executable entity that differs from `NSThread` in that it does not share the memory space of the process that created it. The `NSPipe` allows us a way to read the output of the task created with `NSTask`. Let's see how this all fits together.

Note

Both `NSTask` and `NSPipe` are complex classes that are documented fully by Apple. Be sure to review the documentation for these classes to see all the possible options associated with them.

The `-init` method in [Listing 13.2](#) does little more than initialize our instance variables.

Listing 13.2 MyTerminalController -init Method in MyTerminalController.m

```
- (id) init
{
    [super init];
    if (self) {
        m_pingIsRunning = FALSE;
    }
}
```

```

        m_pingPipe = nil;
        m_pingTask = nil;
    }
    return self;
}

```

The `-dealloc` method in [Listing 13.3](#) merely releases those instance variables that need it. These instance variables will be used when we discuss ping, but first, let's look at the simpler uptime.

Listing 13.3 MyTerminalController `-dealloc` Method in MyTerminalController.m

```

- (void) dealloc
{
    if (m_pingPipe) { [m_pingPipe release]; m_pingPipe = nil; }
    if (m_pingTask) { [m_pingTask release]; m_pingTask = nil; }
    [super dealloc];
}

```

Uptime

[Listing 13.4](#) shows the `-uptime:` method that is called when the Uptime button is pressed in the main application window.

Listing 13.4 MyTerminalController `-uptime:` Method in MyTerminalController.m

```

/*
    uptime is a simple, fast command. It only
    returns one line of text so we quickly
    are able to execute it and read the results
    from the pipe. No need for fancy threads
    or multiple reads here.
*/
- (IBAction)uptime: (id) sender
{
    // Allocate the task to execute and the pipe to send the output to
    NSTask          *theTask = [[NSTask alloc] init];
    NSPipe          *thePipe = [[NSPipe alloc] init];

    // Get the file handle from the pipe (assumes thePipe was allocated!)
    NSFileHandle   *theFileHandle = [thePipe fileHandleForReading];

    // Tell the task what command (program) to execute
    [theTask setLaunchPath: @"/usr/bin/uptime"];

    // Set thePipe as the standard output so we can see the results
    [theTask setStandardOutput: thePipe];

    // Launch the task
    [theTask launch];

    // Wait until the task exits, we know uptime exits immediately
    [theTask waitUntilExit];
}

```

```

// Verify that the program completed without error
if ([theTask terminationStatus] == 0) {
    NSString *theString;

    // Extract a string from the data returned in the pipe's file handle
    theString = [[NSString alloc]
        initWithData:[theFileHandle readDataToEndOfFile]
        encoding:NSUTF8StringEncoding];

    // Set the text field to the value of the string
    [m_textField setStringValue:theString];

    // Release what we create, theFileHandle
    // is automatically released by thePipe
    [theString release];
} else {
    // Set the text field to the value of the error
    [m_textField setIntValue:[theTask terminationStatus]];
}

[thePipe release];
[theTask release];
}

```

The first thing you will note is that the method allocates local `NSTask` and `NSPipe` variables. Because the uptime program is so fast and straightforward, all the work will happen within this method. There is no need to work asynchronously with uptime. After allocating the `NSPipe`, we call its `-fileHandleForReading` method to return an `NSFileHandle` that we can then use to read the data returned by uptime.

Next, we must call `NSTask`'s `-setLaunchPath:` method to set the path to the executable we want to launch. Uptime exists in `/usr/bin/uptime`, so we pass that string as the argument to the method. We then call `NSTask`'s `-setStandardOutput:` method to ensure that the output is written to the `NSPipe` allocated earlier. At this point, we are ready to `-launch` the `NSTask`.

Note

You might also consider capturing `STDERR` using `NSTask`'s `-setStandardError:` to catch any reported errors.

Because uptime executes so quickly, we merely call `NSTask`'s `-waitUntilExit` method to wait for the task to finish. This method will suspend our program until uptime has terminated. Once this method returns, we check the `-terminationStatus` of the `NSTask` to ensure that the program completed without error. Assuming that all is well, we `+alloc` an `NSString` to display the data returned by the `NSFileHandle -readDataToEndOfFile` method, and then release the `NSString`. If there was an error, we simply display the `-terminationStatus` as an integer instead.

Note

We could have created that `NSString` differently. We could've made it `-autorelease` to avoid

the `-release` method call. However, if you know you are done with an object, it's better to `-release` it directly than to have it be destroyed later via `-autorelease`.

Finally, we release both the `NSPipe` and the `NSTask`. The `NSPipe` will release the `NSFileHandle` that we received a reference to earlier.

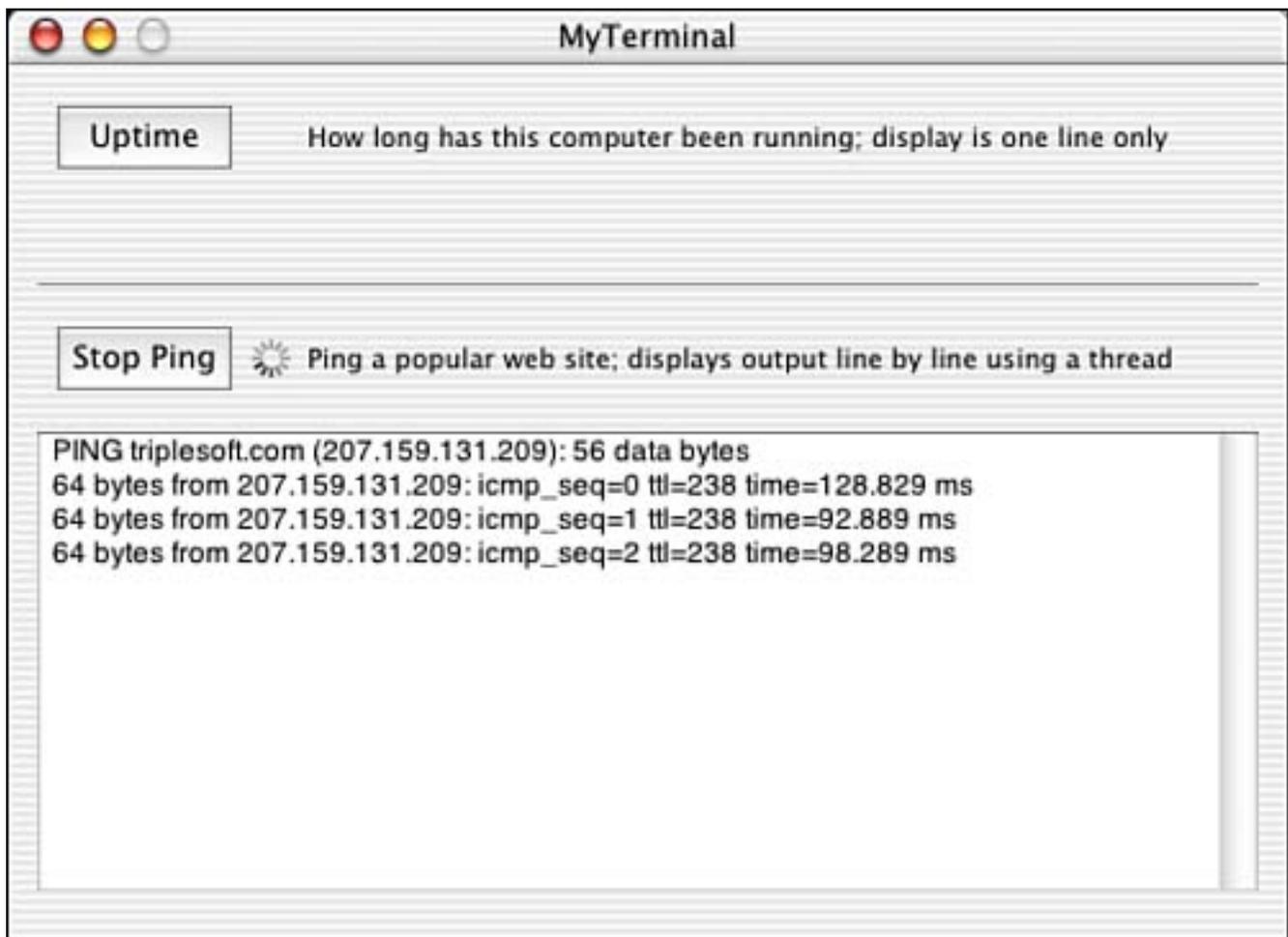
As you can see, it's not too complex to launch a task and read its output.

Ping

Ping is a bit more complex than uptime. Whereas uptime returned within a split second, ping accesses the network and returns data over time. Because of this, we'll choose to use an `NSThread` to read the data from the ping's `NSPipe`. Therefore, we have two methods to look at—one is the `IBAction, -ping:` that is called when the user presses the Start Ping button. The second is the `-displayPingData:` thread method that is executed to watch ping's `NSPipe` (`NSFileHandle` of the `NSPipe` actually) for data and ultimately cleanup when the user presses the Stop Ping button or the ping program terminates on its own.

Note that once you press the Start Ping button, the name changes to Stop Ping while ping is running. [Figure 13.10](#) shows this phenomenon. If the ping task is executing, the button will read Stop Ping.

Figure 13.10. The MyTerminal application in mid ping.



Listing 13.5 shows the `-ping:` method. This function behaves differently depending on the state of the `m_pingIsRunning` instance variable. Remember that the same physical button functions as a Stop Ping button when ping is running. Therefore, if ping is currently running, the method simply sets `m_pingIsRunning` to `FALSE` and disables said button. The value of `m_pingIsRunning` will be picked up in the `-displayPingData:` thread method below as well.

Listing 13.5 MyTerminalController `-ping:` Method in MyTerminalController.m

```

/*
 ping is called when the Start Ping button is pressed.
 It is also called to a ping in progress.
 Essentially, this function sets things up for a
 separate thread that handles reading the data.
 When the users presses the Stop Ping button, the function sets a flag that
 triggers the thread to deallocate the task and stop the ping.
 */
- (IBAction) ping: (id) sender
{
    if (m_pingIsRunning) {
        // If we are currently running and this is called, we want to stop,
        // so set a flag...
        m_pingIsRunning = FALSE;
        // ...and disable the button so it can't be clicked again until we
        // have terminated the ping
        [m_pingButton setEnabled:NO];
    }
}

```

```

} else {
    NSFileHandle *theFileHandle;

    // Otherwise we are currently not pinging so we want to start...
    // allocate a task, a pipe and the file handle
    m_pingTask = [[NSTask alloc] init];
    m_pingPipe = [[NSPipe alloc] init];
    theFileHandle = [m_pingPipe fileHandleForReading];

    if (m_pingTask && m_pingPipe && theFileHandle) {
        // If we get this far we are pretty safe so we set the
        // global flag that we are pinging...
        m_pingIsRunning = TRUE;
        // ...and begin some animation for the user to see activity
        // on the screen
        [m_pingIndicator startAnimation:self];

        // Tell the task what command (program) to execute
        [m_pingTask setLaunchPath:@" /sbin/ping"];

        // Pass some arguments to the program, in this case the domain
        // name to ping 5 times
        [m_pingTask setArguments:[NSArray
            arrayWithObjects:@" -c 5", @"triplesoft.com", nil]];

        // Set m_pingPipe as the standard output so we can see the results
        [m_pingTask setStandardOutput:m_pingPipe];

        // Launch the task
        [m_pingTask launch];

        // Clear the text we placed in the text view in awakeFromNib
        [m_textView setString:@""];

        // Create the thread that will handle reading data from ping
        // and pass theFileHandle from thePipe
        [NSThread detachNewThreadSelector:@selector(displayPingData:)
            toTarget:self withObject:theFileHandle];

        // Change the title of the button to Stop Ping to reflect the
        // change in state
        [m_pingButton setTitle:@"Stop Ping"];
    } else {
        // If there was an error, tell the user
        if (m_pingPipe) { [m_pingPipe release]; m_pingPipe = nil; }
        if (m_pingTask) { [m_pingTask release]; m_pingTask = nil; }
        NSRunAlertPanel(@"MyTerminal",
            @"An error occurred trying to allocate the task, pipe or file handle.",
            @"OK", @"", @"");
    }
}
}
}

```

If ping is not currently running, this method is responsible for kicking it off. In that case, the first thing it must do is `+alloc` and `-init` the `NSTask` and `NSPipe` instance variables of `MyTerminalController`. These are made instance variables because they are used in the thread method below as well. Note that we also

access the `NSPipe's -fileHandleForReading` method as we did in the uptime example.

Note

Note that you could also use `NSThread's -threadDictionary` method to access the thread's dictionary (that we are about to create) and store the `NSTask`, `NSPipe`, and `NSFileHandle` there.

Assuming that we have everything allocated and in order, we continue by setting `m_pingIsRunning` to `TRUE`. We also animate the progress indicator so that the user knows something is happening. `NSTask's -setLaunchPath:` is called to point to `/sbin/ping`. We then make use of its `-setArguments:` method to assign an array of arguments to be passed to ping. Although many arguments are available, we are passing in the `-c` argument to tell ping to ping the host `triplesoft.com` five times before automatically terminating. The user can still stop the ping before it completes, but this will limit the time the program will run.

We then call `NSTask's -setStandardOutput:` method to ensure that the output is written to the `NSPipe` allocated earlier. At this point, we are ready to `-launch` the `NSTask`. Just as we launch the task, we also clear the output text in the `NSTextView` so that we start with a fresh slate on the user interface side of the equation.

Next, we create and detach the thread that will read and display the output from ping to the user. Note that we are creating the `-displayPingData:` method of `self` as the thread. Note also that the `withObject:` parameter is `theFileHandle`. This allows easy access to the `NSFileHandle` written to by the ping task. As you will see shortly, the `NSFileHandle` is passed directly to the thread. We then set the Start Ping button to Stop Ping.

Note the last bit of code in this method is the `else` to handle the case in which the `NSPipe` or `NSTask` were not allocated properly. Just a little cleanup and an error is displayed should anything strange happen.

Now that the task has been set up and kicked off, let's take a look at the thread that maintains it.

[Listing 13.6](#) contains the `-displayPingData:` thread method. This method begins execution the moment it is created in `-ping:`.

Listing 13.6 MyTerminalController -displayPingData: Method in MyTerminalController.m

```
/*
 * This is the thread that handles reading the data from the ping program
 */
- (void)displayPingData: (NSFileHandle*) theFileHandle
{
    // Threads are responsible for manage their own autorelease pools
    NSAutoreleasePool *thePool = [[NSAutoreleasePool alloc] init];

    // While the flag is set (the user has yet to tell us to stop pinging)
    while (m_pingIsRunning) {
```

```

// Read in any available data from the
// file handle passed into the thread
NSData *theData = [theFileHandle availableData];

// If there is data...
if ([theData length]) {
    // Extract a string from the data
    // returned in the pipe's file handle
    NSString *theString = [[NSString alloc]
        initWithData:theData encoding:NSUTF8StringEncoding];

    // Append the text to the end of the
    // text view and scroll it into view
    NSRange theEnd = NSMakeRange([m_textView string] length, 0);
    [m_textView replaceCharactersInRange:theEnd withString:theString];
    theEnd.location += [theString length];
    [m_textView scrollRangeToVisible:theEnd];

    // Release the string
    [theString release];
}

// Sleep this thread for 100ms so as
// to allow other threads time (ie: UI)
[NSThread sleepUntilDate:[NSDate dateWithTimeIntervalSinceNow:(
    NSTimeInterval)0.1]];

// Check if the task has completed and the data is gone
if (([m_pingTask isRunning] == NO) && ([theData length] == 0))
    m_pingIsRunning = NO;
}

// Once the flag has been set to FALSE, we need to clean things up...

// Terminate the ping task and wait for it to exit
[m_pingTask terminate];
[m_pingTask waitUntilExit];

// Check the termination status of ping,
// 15 or 0 means it exited successfully
// 15 = user cancelled, 0 = normal termination
if (([m_pingTask terminationStatus] != 15) &&
    ([m_pingTask terminationStatus] != 0))
    NSRunAlertPanel(@"MyTerminal ",
        @"An error occurred trying to quit the task. (%d)",
        @"OK", @"", @"", [m_pingTask terminationStatus]);

// Release the pipe and task
if (m_pingPipe) { [m_pingPipe release]; m_pingPipe = nil; }
if (m_pingTask) { [m_pingTask release]; m_pingTask = nil; }

// Update the UI
[m_pingIndicator stopAnimation:self];
[m_pingButton setEnabled:YES];
[m_pingButton setTitle:@"Start Ping"];

```

```
// Release the autoreleasepool  
[thePool release];
```

```
// Exit this thread  
[NSThread exit];
```

```
}
```

First, the method allocates its `NSAutoreleasePool` since we've learned that threads are responsible for their own `autorelease` pools. Then, as long as the `m_pingIsRunning` flag is set, the thread loops looking for available data being passed back from ping. If the `NSFileHandle -availableData` method returns data, verified with the `-length` method, it is extracted into a string and appended to the `NSTextView` in the main application window.

The thread is then put to sleep using `NSThread's +sleepUntilDate:` class method to conserve CPU usage. When the thread wakes up, it checks to see if the ping task is still running using the `NSTask` method `-isRunning`—this is called polling. It also checks to see if any data is waiting to be processed. This allows us to verify whether the task completed on its own. If ping is not running and there is no data, the task stopped on its own and we have already read in all there is to read, so we set the `m_pingIsRunning` flag to `NO`. Otherwise, we will continue to loop to pick up any remaining data, such as the final summary that ping outputs.

Note

Here's a few words about polling. Why do I poll? I'm originally a Macintosh programmer. Macintosh programmers poll; it's what we've done for as long as I can remember—it's almost all we know. However, other ways can conserve CPU resources and deserve a mention.

By using tools such as mach port messaging, `NSNotification`, and delegate messages to notify software of important state changes, you can avoid polling. `NSFileHandle`, for example, implements a notification named `NSFileHandleDataAvailableNotification`. By choosing to become an observer to this notification, you will be alerted when data is available to be read, as opposed to polling for the data.

See if you can retrofit this project to support notifications; see the "[Try This](#)" section for another hint as to how to do this.

At some point, the task will either exit on its own, after five pings, or the user will press the Stop Ping button. Either way, the `m_pingIsRunning` flag will be set to `NO`. When this occurs, the loop will exit and the thread will begin to clean things up. First, it will attempt to `-terminate` the task and `-waitUntilExit`. If the task completed on its own, these two methods will return immediately because the task will have already terminated.

Next, the `-terminationStatus` of the task will be checked to see how it was terminated. A value of `15` means that the user cancelled the task. A value of `0` means that the task completed on its own. Anything other than this means that some type of error occurred. We then `-release` the `NSPipe` and `NSTask`, update the user interface, `-release` the `NSAutoreleasePool`, and `-exit` the `NSThread`.

By using a thread, it's relatively easy to read the output from any command-line program that returns data over time. Be sure to read [Chapter 12](#) for more information on threads.

Try This

Here are some ideas to expand the project in this chapter:

Add a view that plots the server load, as a bar chart, using the data returned by uptime beyond the 15-minute limit. Schedule an `NSTimer` to run the uptime program once every minute and keep track of the data returned to plot it for the user in a simple graph.

Add the ability for the user to set various ping options within the application user interface. You can type `man ping` in the terminal to see all the options and their potential values.

Pick another command-line–interface program and implement a graphical user interface for it as I've done here. Can't think of one? Try `users`, `date`, `man`, `cal`, `du`, or `who`.

Rewrite the ping program to work asynchronously without using `NSThread`. (Hint: use `NSFileHandle's` `readInBackgroundAndNotify` method to get data out of `STDERR` and `STDOUT` as it arrives.)

Conclusion

You've learned a lot in this chapter. Using `NSTask` and `NSPipe`, you can leverage the power of proven applications that have existed for ages from within your Cocoa applications. Be sure to review the details of the `NSTask` and `NSPipe` classes because many options are associated with each of them. We've only scratched the surface in this chapter, but I hope that it has given you a spark to learn more about this powerful technique.

Chapter 14. XML-RPC

"Does distributed computing have to be any harder than this? I don't think so."

—Byte Magazine

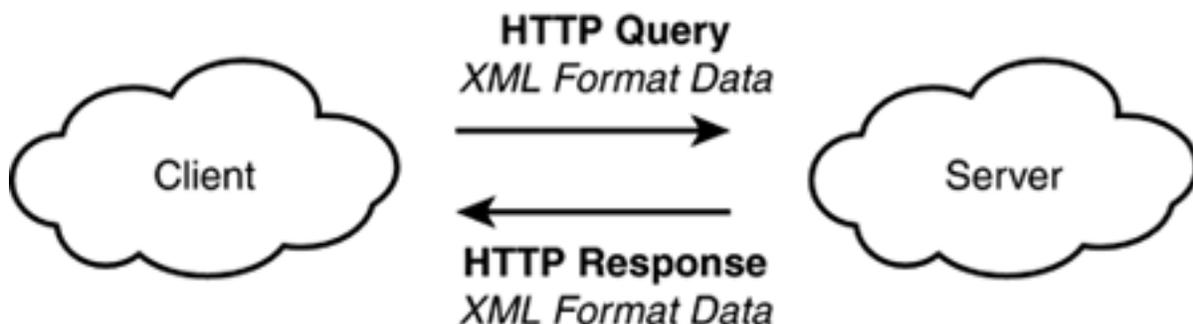
Every once in a while, something comes along that changes the way you think about work. XML-RPC is one of the coolest technologies I've explored in a long time. XML-RPC is a specification to allow software on one computer to call remote procedures (RPCs) on another computer, regardless of platform. The transport mechanism is HTTP, and the encoding mechanism is XML.

XML-RPC was created by Dave Winer of UserLand Software, Inc. and is a simpler alternative to using SOAP, discussed in [Chapter 15](#), "SOAP." Both are considered "Web Services"—XML-RPC being the first. Although both enable complex data structures to be transmitted, processed, and responded to, using XML-RPC is just a tad simpler. XML-RPC is an excellent implementation to base your client/server project because there is no fluff. It does what it does very well.

XML-RPC can be implemented in any of a number of programming languages on just about every available platform. You can choose C/C++ (including Carbon), Frontier, Java, Lisp, Objective-C (including Cocoa), Perl, PHP, Python, BASIC, Tcl, and many more. This offers a lot of flexibility to the programmer looking for a robust solution with minimal limitations.

Both XML-RPC and SOAP are useful when you need to implement a data-based client/server architecture. Although XML-RPC and SOAP are "Web Services", normally they will not return fully formatted HTML, but instead just the raw data. Remember, you don't know what format your client might be—it might be a Web browser, it might be a native application with a Mac OS X or Windows GUI, or it might be a cell phone. Therefore, your server should return data only, leaving the formatting of the data to the client. This is a perfect use for XML. [Figure 14.1](#) shows this concept.

Figure 14.1. A client and server via XML-RPC.



The official XML-RPC Web site can be found at <http://www.xmlrpc.org/>. It has extremely detailed information on the protocol that we will use in this chapter.

The Result

The project in this chapter is the same project we will use in [Chapter 15](#). One thing I found while learning about Web services is that it was very difficult to find concrete examples that accepted multiple parameters in and delivered multiple parameters out—like the real-world. By showing a single example using XML-RPC and SOAP as well as both PHP and Objective-C, I hope this will take some of the mystery out of these exciting protocols.

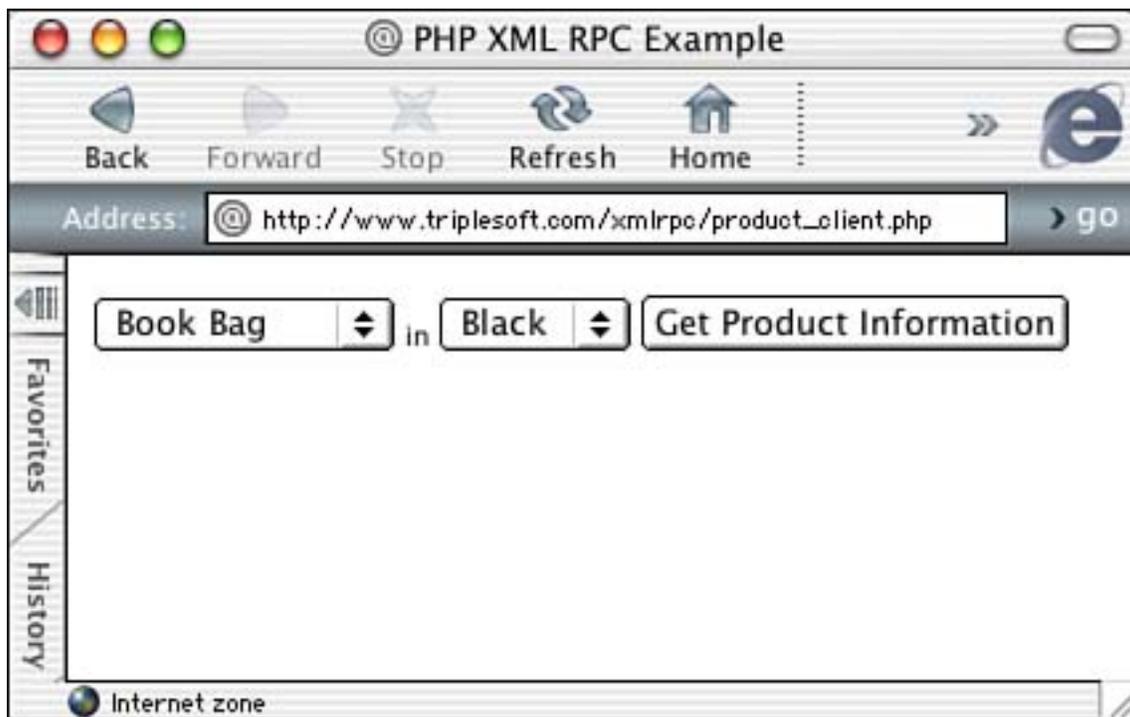
This project shows how to create an XML-RPC server written in PHP and running under the Apache Web server. We then create a PHP client as well as Cocoa Objective-C client. This shows that one server can handle multiple clients using multiple implementation mechanisms. Although the Cocoa client will only run under Mac OS X, the PHP client can be used on any computer with a Web browser.

My example is a product information system. We sell many different styles of bags in a rainbow of colors—everything from book bags to handbags to grocery bags and more. Using one of our clients, the user simply picks a type of bag and a color, and the server returns the price and number of bags currently in stock that matches those criteria. For the purposes of this demo, any bag in the color brown is out of stock, whereas all other colors return a random stock value. Normally, you would pull this information from a database, but this is left as an exercise to the reader.

The PHP client is served via Apache and is essentially a Web form. The client accepts data from the user, sends the request to the server, formats the resulting data appropriately, and displays it in a Web browser.

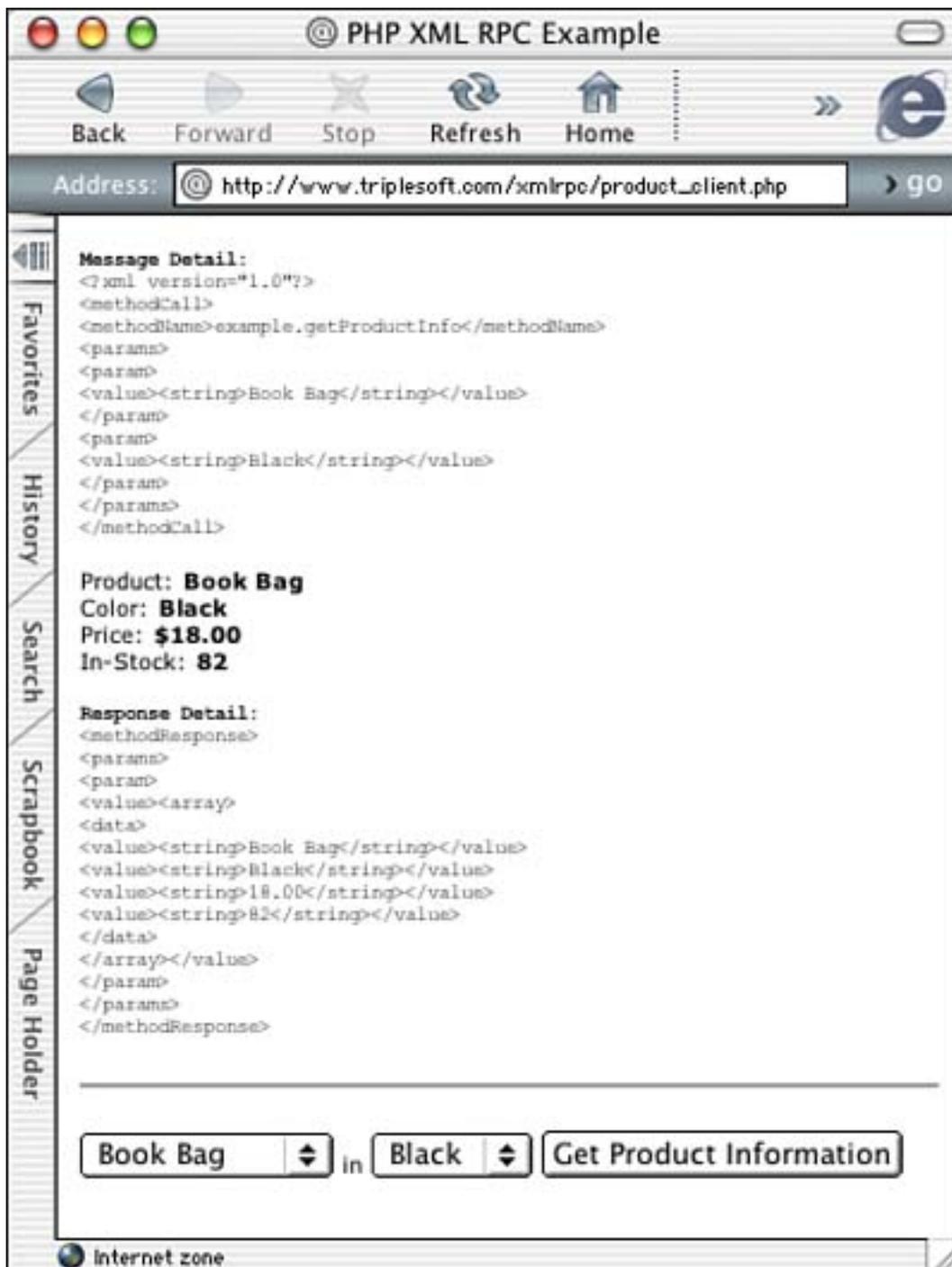
[Figure 14.2](#) shows the PHP client running in Internet Explorer under Mac OS X although it could run in any modern Web browser. The user is presented with a form that allows her to choose which type of bag and which color she wants price and stock information for. Once chosen, she clicks the Get Product Information button, which passes the form data on to the server.

Figure 14.2. The PHP XML-RPC client before submitting a query.



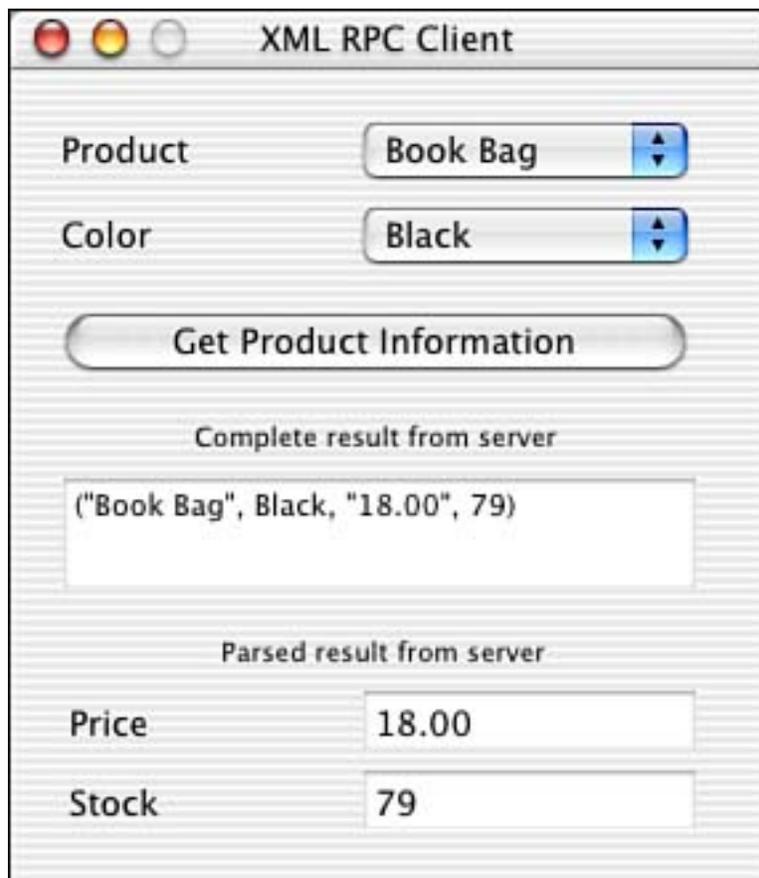
Once the server receives and processes the data, it returns the resulting data (or an error) to the client. The client is then responsible for displaying the results to the user. [Figure 14.3](#) shows our client displaying the original XML-format query, or message, the formatted results, and then the details of the response from the server that were parsed to produce the formatted results. Normally you wouldn't display the XML-formatted information to the user, but it is very helpful to see the complete picture when you're learning something new.

Figure 14.3. The PHP XML-RPC client after submitting a query and receiving the results.



The Cocoa client, shown in [Figure 14.4](#), works similarly in that it collects data from the user and then contacts the server to perform the price and stock lookup. However, after it receives and parses the response, it displays it within its own window differently than the previous Web browser example. It still uses HTTP as the transport mechanism, which is standard for XML-RPC, but does not display the results using HTML.

Figure 14.4. The Cocoa XML-RPC client after submitting a query and receiving the results.



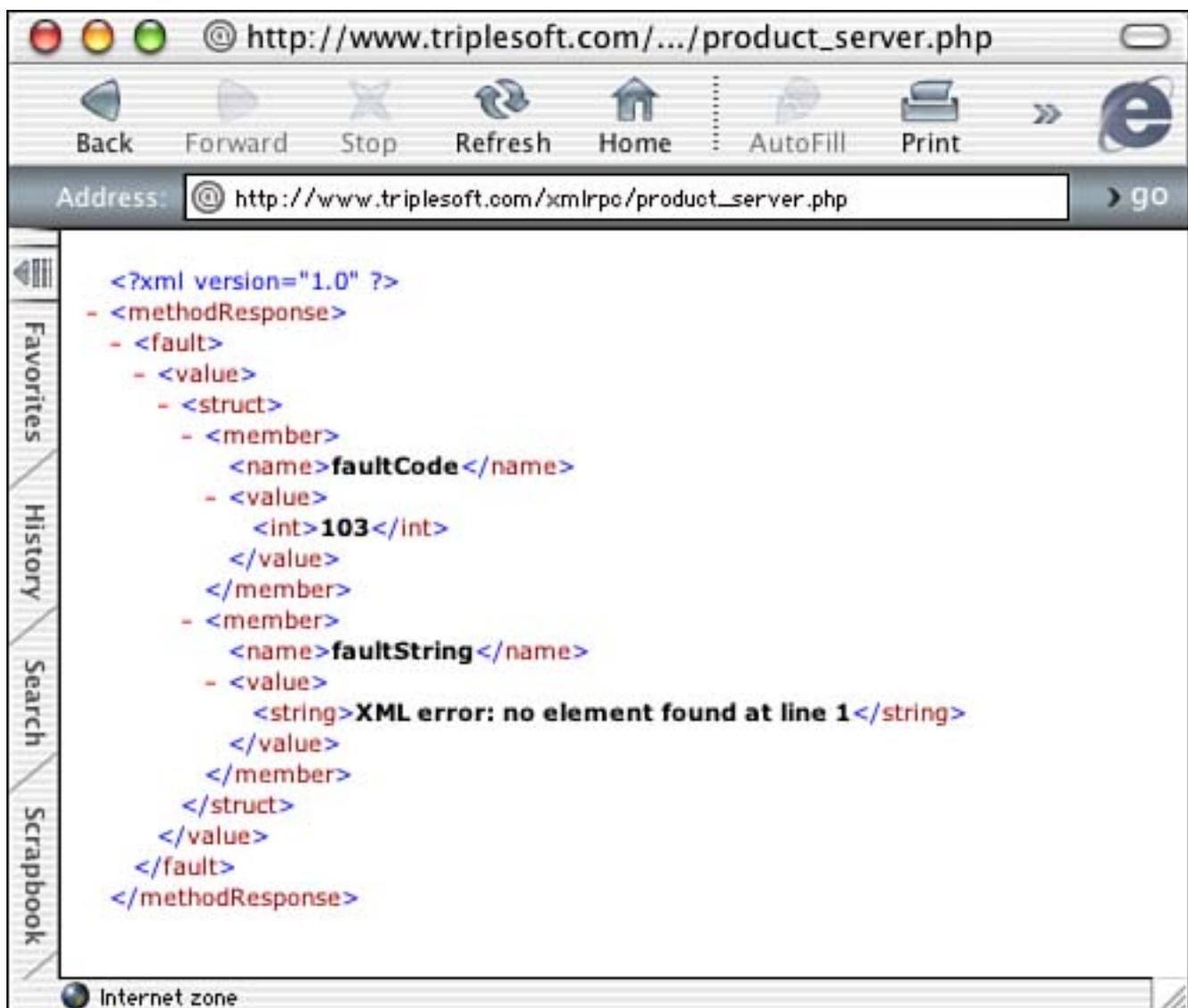
Let's look at how this is all put together.

[\[Team LiB \]](#)

The PHP Server

First, we will look at the server side. As mentioned, the server is written in PHP but can be written in almost any language. We chose PHP because it is reliable, straightforward, and powerful. Our `product_server.php` file is served by the Apache Web server running under Mac OS X just as any standard PHP file would be. The client accesses it via a standard HTTP URL. In this case, we have placed the file at `http://www.triplesoft.com/xmlrpc/product_server.php`. However, typing that URL into your Web browser will simply produce an error because XML-RPC servers require specifically formatted information passed by the client. [Figure 14.5](#) shows the fault information returned by the server.

Figure 14.5. The PHP XML-RPC server being called directly from the Web browser.



An important thing to note is that both the PHP server and client require the use of a third-party PHP library by Edd Dumbill. The `XML-RPC for PHP` library and detailed information on its use can be found at <http://www.usefulinc.com/>. This library contains two include files, `xmlrpc.inc` and `xmlrpcs.inc`, that are easily included in your PHP source. The distribution also contains documentation and examples as well as

licensing information.

Let's look at the complete PHP source code of the server and then I will explain it in sections.

[Listing 14.1](#) contains the complete PHP XML-RPC server code. This simple server offers a concrete example of the protocol accepting multiple parameters and returning multiple parameters, which is most likely what you will need in a real-life scenario.

Listing 14.1 product_server.php

```
<?php
include("xmlrpc.inc");
include("xmlrpcs.inc");

// Array of products and prices, in real-life you might use a SQL database
$products = array (
    "Grocery Bag" => "9.00",
    "Book Bag" => "18.00",
    "Hand Bag" => "32.00",
    "Garment Bag" => "48.00",
    "Old Bag" => "0.31"
);

// Signature of this RPC
$products_signature =
    array(array($xmlrpcArray, // Returns a product price and quantity
        $xmlrpcString, // Accepts a product name
        $xmlrpcString)); // Accepts a color

// Documentation string of this RPC
$products_docstring = "When passed valid product info, more info is returned.";

// RPC Main
function get_product_info($input)
{
    global $xmlrpcerruser, $products;
    $err = "";

    // Get the parameters
    $param0 = $input->getParam(0);
    $param1 = $input->getParam(1);

    // Verify value and type
    if ((isset($param0) && ($param0->scalartyp() == "string")) &&
        (isset($param1) && ($param1->scalartyp() == "string"))) {

        // Extract parameter values
        $product_name = $param0->scalerval();
        $product_color = $param1->scalerval();

        // Attempt to find the product and price
        foreach ($products as $key => $element) {
            if ($key == $product_name) {
                $product_price = $element;
            }
        }
    }
}
```

```

    }
}

// For this demo, all Browns are out of stock and all other colors
// produce a random in-stock quantity - probably wouldn't work like
// this in real-life, you would probably use a SQL database instead
if ($product_color == "Brown") {
    $product_stock = 0;
} else {
    $product_stock = rand(1, 100);
}

// If not found, report error
if (!isset($product_price)) {
    $err = "No product named '" . $product_name . "' found.";
}

} else {
    $err = "Two string parameters (product name and color) are required.";
}

// If an error was generated, return it, otherwise return the proper data
if ($err) {
    return new xmlrpcresp(0, $xmlrpcerruser, $err);
} else {
    // Create an array and fill it with the data to return
    $result_array = new xmlrpcval(array(), "array");
    $result_array->addScalar($product_name, "string");
    $result_array->addScalar($product_color, "string");
    $result_array->addScalar($product_price, "string");
    $result_array->addScalar($product_stock, "string");
    return new xmlrpcresp($result_array);
}
}

// Create the server
$server = new xmlrpc_server(
    array("example.getProductInfo" =>
        array("function" => "get_product_info",
            "signature" => $products_signature,
            "docstring" => $products_docstring),
    )
);
?>

```

Let's look at the server code by section. First, in [Listing 14.2](#), you will notice the two include statements for the `xmlrpc.inc` and `xmlrpcs.inc` files. Servers will make use of both of these files; however, clients will only use `xmlrpc.inc` in most cases.

Listing 14.2 Initial Steps in `product_server.php`

```

include("xmlrpc.inc");
include("xmlrpcs.inc");

```

```

// Array of products and prices, in real-life you might use a SQL database
$products = array (
    "Grocery Bag" => "9.00",
    "Book Bag" => "18.00",
    "Hand Bag" => "32.00",
    "Garment Bag" => "48.00",
    "Old Bag" => "0.31"
);

// Signature of this RPC
$products_signature =
    array(array($xmlrpcArray, // Returns a product price and quantity
        $xmlrpcString, // Accepts a product name
        $xmlrpcString)); // Accepts a color

// Documentation string of this RPC
$products_docstring = "When passed valid product info, more info is returned.";

```

Next, we create an array of products with prices, appropriately named `$products`. In a real-life scenario, you will most likely use some sort of database for this information. MySQL is a common choice when using PHP and Apache, but other options exist as well.

We initialize the `$products_signature` variable, which contains the signature for the RPC. The signature defines the parameters expected from and returned to the client. The signature is an array that contains a list of the data types in question. The first entry in the array is the return type—in this case, an array of items is returned by the server. Subsequent entries are the individual parameters expected of the client—in this case, a string representing the product name and a string representing the product color. You could just as easily make the second and third parameters part of an array as well.

Last, we initialize the `$products_docstring` variable, which contains the documentation string for the RPC. This is used to describe the RPC to a client in human-readable terms.

The `get_product_info` function in [Listing 14.3](#) is the main and only RPC of this server. Although servers can serve multiple RPCs, only one is implemented in this example. This function does all the work—parses the incoming parameters, performs any calculations or database lookups, and returns the results to the client. This function is automatically called when the server receives the request based on information within the XML-formatted data sent to the server. See [Listing 14.4](#) for an example of the XML sent to the server that triggers this function call.

Listing 14.3 The `get_product_info` Method in `product_server.php`

```

function get_product_info($input)
{
    global $xmlrpcerruser, $products;
    $err = "";

    // Get the parameters
    $param0 = $input->getParam(0);
    $param1 = $input->getParam(1);

    // Verify value and type
    if ((isset($param0) && ($param0->scalartyp() == "string")) &&

```

```

        (isset($param1) && ($param1->scalartyp() == "string"))) {

// Extract parameter values
$product_name = $param0->scal arval ();
$product_color = $param1->scal arval ();

// Attempt to find the product and price
foreach ($products as $key => $element) {
    if ($key == $product_name) {
        $product_price = $element;
    }
}

// For this demo, all Browns are out of stock and all other colors
// produce a random in-stock quantity - probably wouldn't work like
// this in real-life, you would probably use a SQL database instead
if ($product_color == "Brown") {
    $product_stock = 0;
} else {
    $product_stock = rand(1, 100);
}

// If not found, report error
if (!isset($product_price)) {
    $err = "No product named '" . $product_name . "' found.";
}

} else {
    $err = "Two string parameters (product name and color) are required.";
}

// If an error was generated, return it, otherwise return the proper data
if ($err) {
    return new xmlrpcresp(0, $xmlrpcerruser, $err);
} else {
    // Create an array and fill it with the data to return
    $result_array = new xmlrpcval(array(), "array");
    $result_array->addScal ar($product_name, "string");
    $result_array->addScal ar($product_color, "string");
    $result_array->addScal ar($product_price, "string");
    $result_array->addScal ar($product_stock, "string");
    return new xmlrpcresp($result_array);
}
}
}

```

Listing 14.4 The XML-Formatted Data Sent to product_server.php

```

<?xml version="1.0"?>
<methodCall>
  <methodName>example.getProductInfo</methodName>
  <params>
    <param>
      <value><string>Book Bag</string></value>
    </param>
  </params>
</methodCall>

```

```

        <param>
            <val ue><stri ng>Bl ack</stri ng></val ue>
        </param>
    </params>
</met hodCall >

```

Looking at [Listing 14.3](#) again, first we allow access to two global variables—`$xml rpcerruser` from `xml rpc. inc` and the `$products` array we defined previously. `$xml rpcerruser` will contain any error code that is generated during the execution of the server. We also initialize the `$err` string to "".

Next, we verify the parameter types and extract their values using the `getParam` function and `scal artyp` and `scal arval` methods. Remember, parameter `0` is the product name and parameter `1` is the color. The parameters must be sent to the server in that order. Note that [Listing 14.4](#) contains the parameters in the correct (and expected) order. If both parameters do not exist or are not the correct type, we return a human-readable error in `$err`.

After the parameter values have parsed properly, we can then begin to use the data they contain. The first thing to do is peruse the `$products` array using `foreach` for the chosen product. Because we limit the users choices in the client, we should always get a match here. Once found, we set the `$product_pri ce` variable accordingly. If the `$product_pri ce` is not found, which should never happen in our simple example, we return a human-readable error in `$err` a few lines further down.

Next, we figure out the number of the product in the chosen color that is in stock. Remember, for the purposes of this demo, all products chosen in brown are out of stock. All other colors produce a random stock value. In reality, you might pull this information from a SQL database.

Last, if an error was returned, `$err` is set; we return the error using the `xml rpcresp` function in `xml rpc. inc` to create the XML-RPC response. You can see an example of an error returned in [Figure 14.5](#).

However, if all is well, we create our result array. Remember, our server returns an array of data to the client. We first initialize the `$resul t_array` variable using the `xml rpcval` function in `xml rpc. inc` to create an `xml rpcval` object. Once created, we simply call the `addScal ar` method of that object to add the result data in the expected order. You can see that we return four strings. The product name and color are the same items passed to the server—we return them for convenience to the client because HTTP is a stateless protocol. The product price and stock are calculated by our server. We then finish it by passing the array to the `xml rpcresp` function in `xml r pc. inc` to create the XML-RPC response. [Listing 14.5](#) shows an example of the data returned from the server, including the parameters in the correct (and expected) order.

Listing 14.5 The XML-Formatted Data Returned from `product_server. php`

```

<met hodResponse>
    <params>
        <param>
            <val ue><array><dat a>
                <val ue><stri ng>Book Bag</stri ng></val ue>
                <val ue><stri ng>Bl ack</stri ng></val ue>
                <val ue><stri ng>18. 00</stri ng></val ue>
                <val ue><stri ng>82</stri ng></val ue>
            </dat a></array></val ue>
        </param>
    </params>

```

</methodResponse>

The last thing to do, actually the first thing that must occur when the server is called, is to create the server object itself. [Listing 14.6](#) performs this task.

Listing 14.6 Creating the Server in `product_server.php`

```
// Create the server
$server = new xmlrpc_server(
    array("example.getProductInfo" =>
        array("function" => "get_product_info",
            "signature" => $products_signature,
            "docstring" => $products_docstring),
    );
```

When the URL of the server is loaded, this code must run to prepare the server to accept incoming requests. Essentially, the `xmlrpc_server` function in `xmlrpc.inc` creates the server object. This function takes multiple parameters that define not only the names of the RPCs that the server implements—in this case, only `example.getProductInfo`—but also detailed information about each of those methods. In the example, this includes the actual name of the function in code, `get_product_info`; the signature, `$products_signature`; and the documentation string, `$products_docstring`.

Essentially, we register the `get_product_info` function as a method of the server object with the method name `example.getProductInfo`. Although we name our method `example.getProductInfo`, you might consider using the reverse domain method of naming your implementation—something like `com.mydomainname.myprojectname.mymethodname`.

That, my friends, is the server. Now let's look at the PHP client.

[\[Team LIB \]](#)

◀ PREVIOUS NEXT ▶

The PHP Client

Our first client is also written in PHP, just as the server. As you've seen, however, it could be written in just about any language. Later in this chapter, we will rewrite the client using Objective-C and Cocoa. The `product_client.php` file is served by the Apache Web server running under Mac OS X just as any standard PHP file would be. The client file is what the user will access via her Web browser. In this case, we have placed the file at http://www.triplesoft.com/xmlrpc/product_client.php. Typing that URL in your Web browser will produce the Web page shown in [Figure 14.2](#).

When the user clicks the Get Product Information button in the Web form, the pop-up menu selections are sent to the server as XML-formatted data. The server then processes the selections and returns the price and stock information as previously discussed. [Figure 14.3](#) shows what the user's Web browser looks like after the server responds.

[Listing 14.7](#) contains the complete PHP XML-RPC client code. This simple client offers a concrete example of the protocol passing multiple parameters and parsing multiple parameters in return.

Listing 14.7 `product_client.php`

```
<html >
<head><title>PHP XML RPC Example</title></head>
<body>
<basefont size="2" face="Verdana, Arial ">

<?php
include("xmlrpc.inc");

// Show the results of the previous query
// if the parameter there are results to show
if (($HTTP_POST_VARS["product_name"] != "") &&
    ($HTTP_POST_VARS["product_color"] != "")) {

    // Create the message object using the RPC name, parameter and its type
    $message = new xmlrpcmsg("example.getProductInfo",
        array(new xmlrpcval($HTTP_POST_VARS["product_name"], "string"),
            new xmlrpcval($HTTP_POST_VARS["product_color"], "string")));

    // Display the message detail
    print "<pre><b>Message Detail:<br /></b>" .
        htmlentities($message->serialize()) . "</pre>";

    // Create the client object which points to the server
    // and PHP script to contact
    $client = new xmlrpc_client("/xmlrpc/product_server.php",
        "www.triplesoft.com", 80);

    // Enable debug mode so we see all there is to see coming and going
    // $client->setDebug(1);
    // print "<pre><b>From Server:<br /></b></pre>";
```

```

// Send the message to the server for processing
$response = $client->send($message);

// If no response was returned there was a fatal error
// (no network connection, no server, etc.)
if (!$response) { die("Send failed."); }

// If no error, display the results
if (!$response->faultCode()) {

    // Extract the values from the response
    $result_array = $response->value();
    if ($result_array->arraysize() == 4) {
        $product_name = $result_array->arraymem(0);
        $product_color = $result_array->arraymem(1);
        $product_price = $result_array->arraymem(2);
        $product_stock = $result_array->arraymem(3);

        print "<br />";
        print "<br />Product: <b>" . $product_name->scal arval () . "</b>";
        print "<br />Col or: <b>" . $product_col or->scal arval () . "</b>";
        print "<br />Pri ce: <b>$" . $product_pri ce->scal arval () . "</b>";
        print "<br />In-Stock: <b>" . $product_stock->scal arval () . "</b>";
        print "<br />";

    } else {

        print "<br />";
        print "<br />Incorrect number of items in array. " .
            "Should be 4 but there are <b>" .
            $result_array->arraysi ze() . "</b>.";

        print "<br />";
    }

    // Display the response detail
    print "<pre><b>Response Detail:<br /></b>" .
        htmlentities($response->seri al i ze()) . "</pre>";

} else {
    print "<br />";
    print "<br /><b>Fault Code:</b> " . $response->faul tCode();
    print "<br /><b>Fault String:</b> " . $response->faul tStri ng();
    print "<br />";
}

print "<hr>";

}
?>

```

```

<form action="<?php echo "$PHP_SELF";?>" method="POST">
  <select name="product_name">
    <option label="Book Bag"
      value="Book Bag" selected>Book Bag</option>
    <option label="Garment Bag"
      value="Garment Bag">Garment Bag</option>
  </select>

```

```

        <option label="Grocery Bag"
            value="Grocery Bag">Grocery Bag</option>
        <option label="Hand Bag"
            value="Hand Bag">Hand Bag</option>
        <option label="Old Bag"
            value="Old Bag">Old Bag</option>
    </select>
    in
    <select name="product_color">
        <option label="Black" value="Black" selected>Black</option>
        <option label="Blue" value="Blue">Blue</option>
        <option label="Brown" value="Brown">Brown</option>
        <option label="Green" value="Green">Green</option>
        <option label="Red" value="Red">Red</option>
        <option label="White" value="White">White</option>
    </select>
    <button type="submit" name="Get Product Information"
        value="Get Product Information">Get Product Information</button>
</form>

</body>
</html>

```

Let's look at the client code by section. Unlike our server PHP code, our client PHP code contains both HTML and PHP intermixed. The first section does little more than implement the standard HTML tags required for a Web page (see [Listing 14.8](#)).

Listing 14.8 In the Beginning of `product_client.php`

```

<html >
<head><title>PHP XML RPC Example</title></head>
<body>
<basefont size="2" face="Verdana, Arial">

```

[Listing 14.9](#), however, gets us into the code that handles the XML-RPC meat of our client.

Listing 14.9 In the Middle of `product_client.php`

```

<?php
include("xml_rpc.inc");

// Show the results of the previous query
// if the parameter there are results to show
if (($HTTP_POST_VARS["product_name"] != "") &&
    ($HTTP_POST_VARS["product_color"] != "")) {

    // Create the message object using the RPC name, parameter and its type
    $message = new xmlrpcmsg("example.getProductInfo",
        array(new xmlrpcval($HTTP_POST_VARS["product_name"], "string"),
            new xmlrpcval($HTTP_POST_VARS["product_color"], "string")));

    // Display the message detail
    print "<pre><b>Message Detail:<br /></b>" .

```

```
html entities($message->serialize()) . "</pre>";
```

```
// Create the client object which points to  
// the server and PHP script to contact  
$client = new xmlrpc_client("/xmlrpc/product_server.php",  
    "www.triplesoft.com", 80);
```

```
// Enable debug mode so we see all there is to see coming and going  
// $client->setDebug(1);  
// print "<pre><b>From Server: <br /></b></pre>";
```

```
// Send the message to the server for processing  
$response = $client->send($message);
```

```
// If no response was returned there was a fatal error  
// (no network connection, no server, etc.)  
if (!$response) { die("Send failed."); }
```

```
// If no error, display the results  
if (!$response->faultCode()) {
```

```
    // Extract the values from the response  
    $result_array = $response->value();  
    if ($result_array->arraysize() == 4) {  
        $product_name = $result_array->arraymem(0);  
        $product_color = $result_array->arraymem(1);  
        $product_price = $result_array->arraymem(2);  
        $product_stock = $result_array->arraymem(3);  
  
        print "<br />";  
        print "<br />Product: <b>" . $product_name->scal arval () . "</b>";  
        print "<br />Color: <b>" . $product_color->scal arval () . "</b>";  
        print "<br />Price: <b>$" . $product_price->scal arval () . "</b>";  
        print "<br />In-Stock: <b>" . $product_stock->scal arval () . "</b>";  
        print "<br />";
```

```
    } else {  
  
        print "<br />";  
        print "<br />Incorrect number of items in array. " .  
            "Should be 4 but there are <b>" .  
            $result_array->arraysize() . "</b>.";  
        print "<br />";  
    }  
}
```

```
// Display the response detail  
print "<pre><b>Response Detail: <br /></b>" .  
    html entities($response->serialize()) . "</pre>";
```

```
} else {  
    print "<br />";  
    print "<br /><b>Fault Code: </b>" . $response->faultCode();  
    print "<br /><b>Fault String: </b>" . $response->faultString();  
    print "<br />";  
}
```

```

print "<hr>";
}
?>

```

[Listing 14.9](#) begins by including the `xml rpc. inc` file. We then verify that we have values for both of the `$HTTP_POST_VARS` that we are interested in. If both `product_name` and `product_color` are set, we continue creating the XML-RPC message.

Using the `xml rpcmsg` function in `xml rpc. inc`, we select the name of the RPC method that we intend to call on the remote server: `example.getProductInfo`. We also add the two expected parameters, in the correct order, to the array of parameters within the message: `product_name` and `product_color` as strings. We then display this message in its serialized form for debugging purposes. This is shown in [Figure 14.3](#).

After our message is created, we can then put together the client object using the `xml rpc_client` function in `xml rpc. inc`. This function takes multiple parameters, which ultimately make up the URL to the server—in our case, a PHP script living at `http://www.triplesoft.com/xmlrpc/product_server.php`, on port 80, the default HTTP port. It returns a client object used to send the message to the server as the variable `$client`.

At this point, you can enable debug mode by uncommenting the `setDebug` method of the `$client` object. In either case, we send the message using the `send` method of the `$client` object. This single method handles locating the remote server, transmitting the message to the client via the HTTP protocol, and accepting any resulting data from the server, as the `$response` variable. If there is no response, the function dies with an error message; otherwise, it continues parsing the result.

If the `$response` object's `faultCode` method returns an empty result, there is no error and we can continue with the parsing. Otherwise, we display the `faultCode` and `faultString` to the user as the error message. These are sent from the server; `faultString` returns the `$err` string returned by the server.

Assuming that there is no error, we pull the `$response` object's array of parameters as `$result_array` by calling its `value` method. If the size of the array is anything other than four, there is an error and we display it accordingly. Remember, the server returns four values in the array of results in a specific order. We then use the `arraymem` method of the `$result_array` and `scal arval` method of each array member to display the results back to the user.

That ends the PHP portion of our file.

[Listing 14.10](#) is nothing more than the HTML form portion of our client file. This portion appears each time the page is displayed as seen in [Figures 14.2](#) and [14.3](#). It simply contains the pop-up menus and Get Product Information button and calls back to itself as the form action. Although this example hard-codes the product names and colors into the form, a real-life application might query the server for the list of products and list of available colors. See the ["Try This"](#) section of this chapter for your assignment to do just that!

Listing 14.10 In the End of `product_client.php`

```

<form action="<?php echo "$PHP_SELF"; ?>" method="POST">
  <select name="product_name">
    <option label="Book Bag"

```

```
        value="Book Bag" selected>Book Bag</option>
<option label="Garment Bag"
    value="Garment Bag">Garment Bag</option>
<option label="Grocery Bag"
    value="Grocery Bag">Grocery Bag</option>
<option label="Hand Bag"
    value="Hand Bag">Hand Bag</option>
<option label="Old Bag"
    value="Old Bag">Old Bag</option>
</select>
in
<select name="product_color">
    <option label="Black" value="Black" selected>Black</option>
    <option label="Blue" value="Blue">Blue</option>
    <option label="Brown" value="Brown">Brown</option>
    <option label="Green" value="Green">Green</option>
    <option label="Red" value="Red">Red</option>
    <option label="White" value="White">White</option>
</select>
<button type="submit" name="Get Product Information"
    value="Get Product Information">Get Product Information</button>
</form>

</body>
</html>
```

Now let's see how we can implement this client using Objective-C.

[\[Team LiB \]](#)

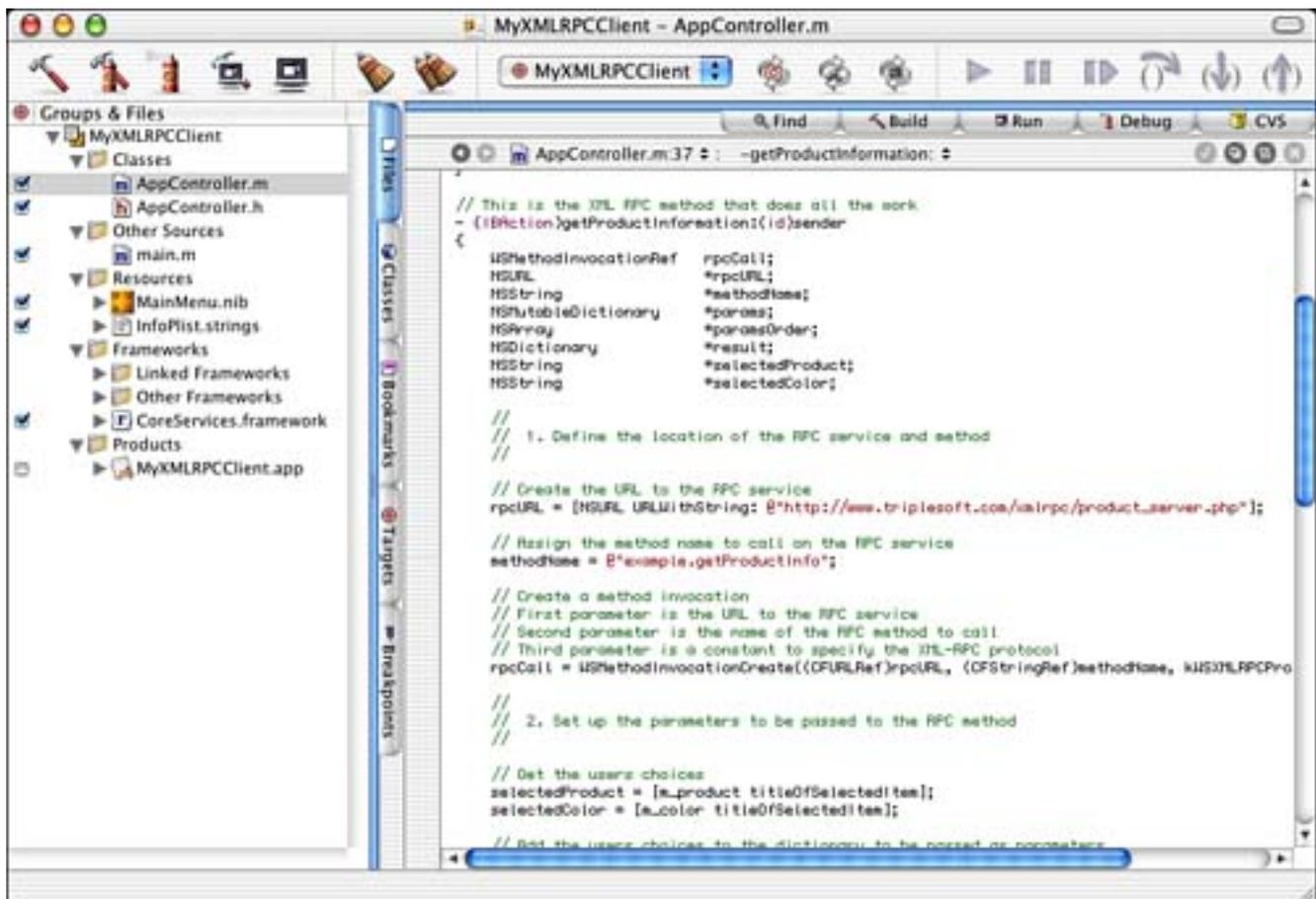
◀ PREVIOUS

NEXT ▶

The Cocoa Client

Figure 14.6 shows the MyXMLRPCClient project in Project Builder. The project is a simple Cocoa application. We use an `AppController` to implement the `-applicationShouldTerminateAfterLastWindowClosed:` delegate method. For the sake of convenience, we also implement the `-getProductInformation:` method in the `AppController` and link it to the Get Product Information button in our main application window shown in Figure 14.4. We could have created a window controller to handle this, but decided against it for the sake of brevity.

Figure 14.6. The MyXMLRPCClient project.



One thing to note is the addition of `CoreServices.framework` in the project. This framework contains functionality that allows us to access Web Services such as XML-RPC and SOAP. You'll see more of what this brings us when we look at the source code later.

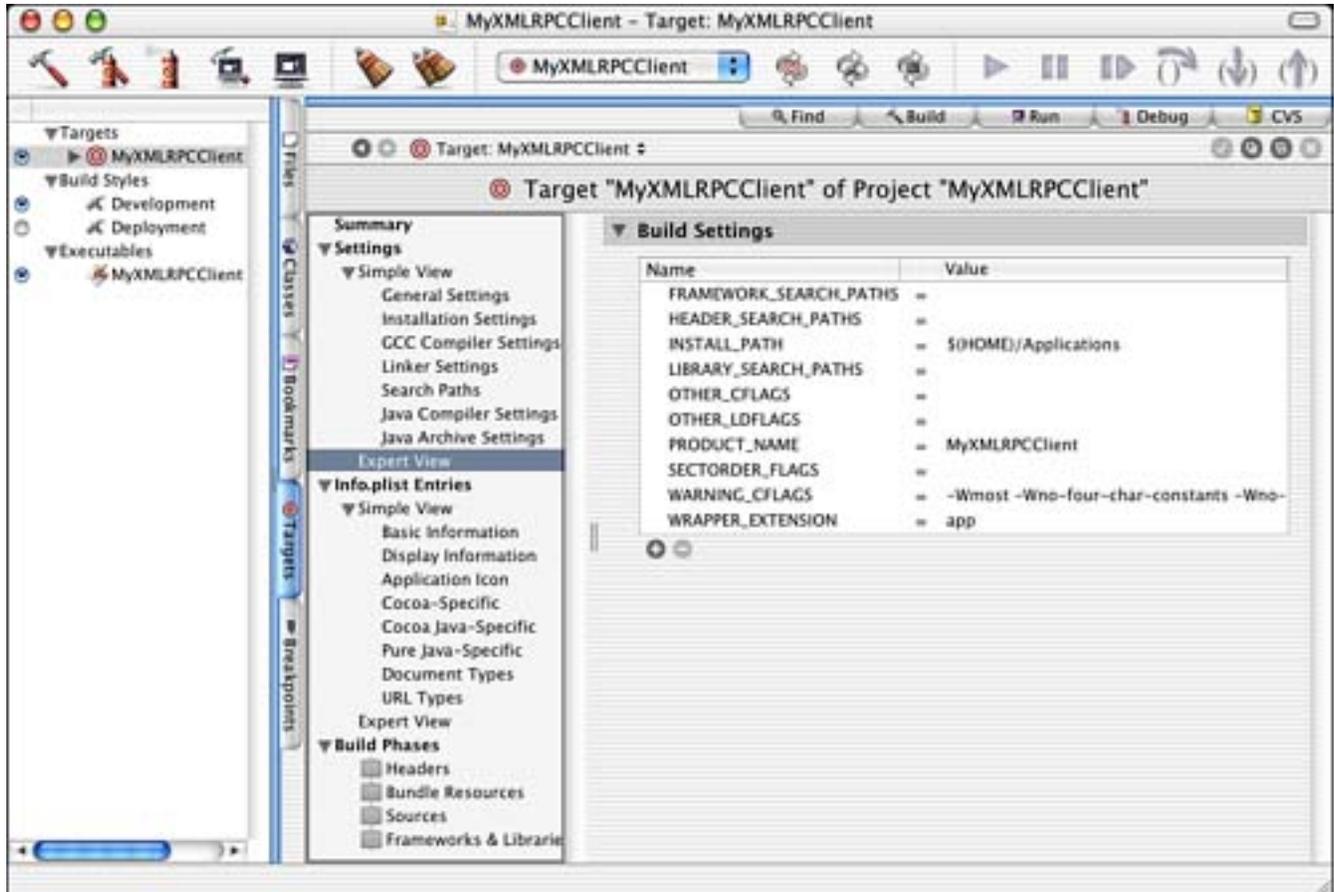
Note

Speaking of frameworks, exploring the `/System/Library/Frameworks/` directory can open the doors to you as an informed programmer. Take a peek, and see what's in there.

The Project Settings

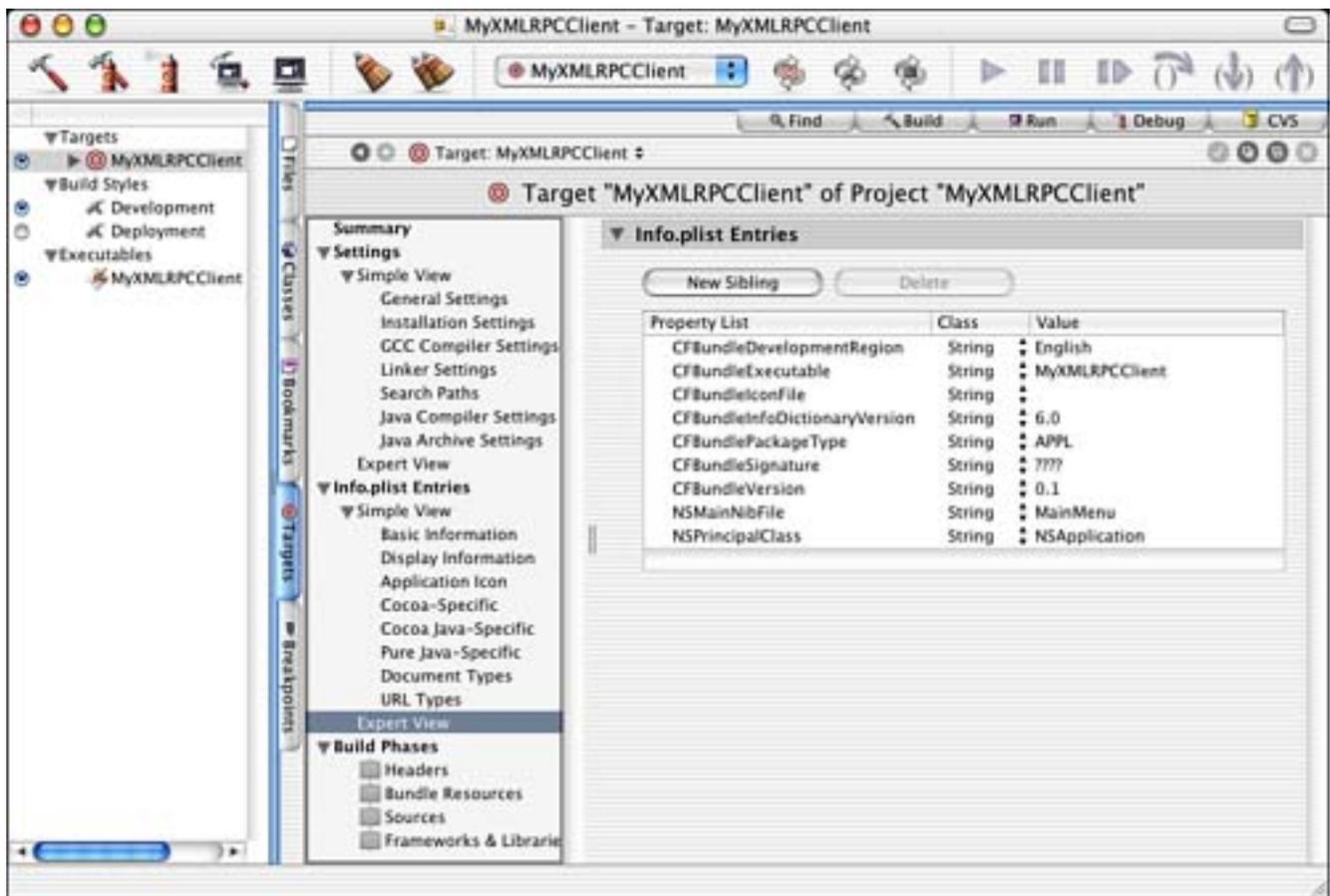
The Target settings in [Figure 14.7](#) are straightforward for a simple Cocoa application.

Figure 14.7. The MyXMLRPCClient Target settings.



The InfoPlist entries in [Figure 14.8](#) are also straightforward. Because this is a simple Cocoa application, there are no surprises here.

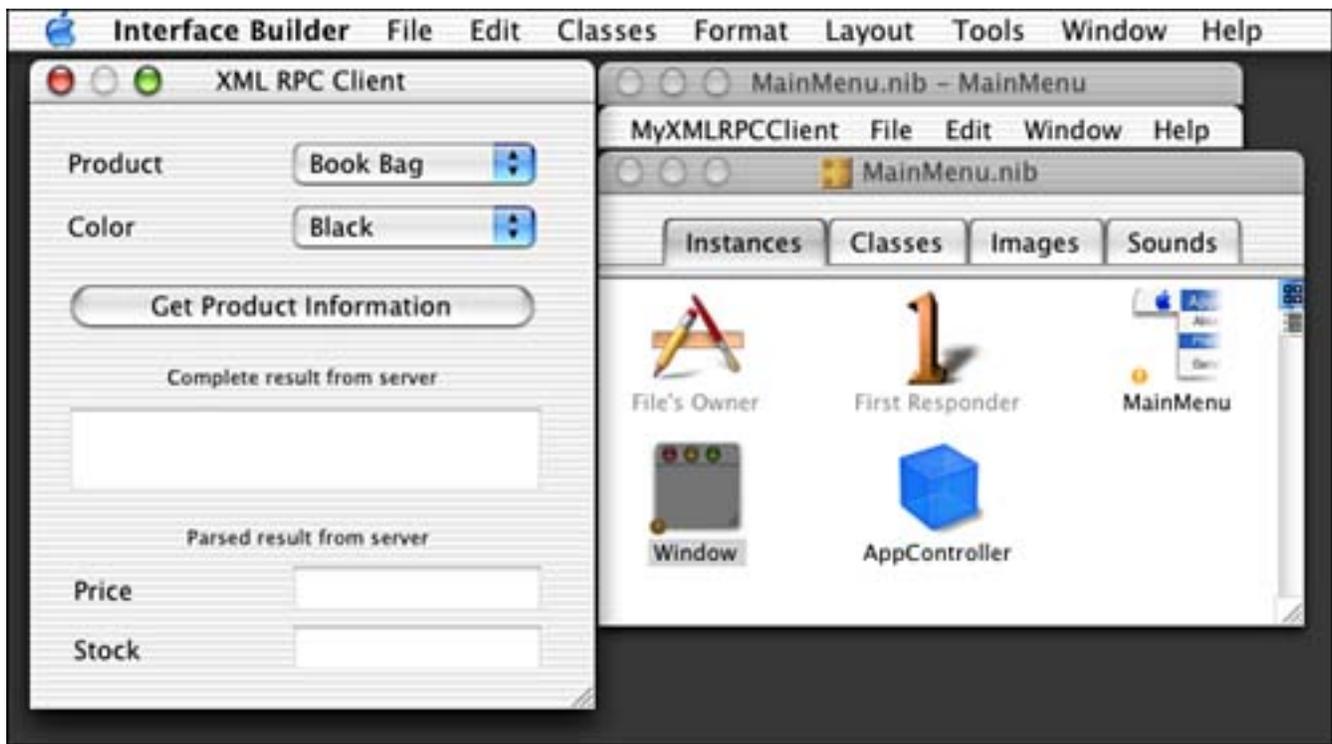
Figure 14.8. The MyXMLRPCClient InfoPlist entries.



The Nib File

The `MainMenu.nib` file is shown in Interface Builder in [Figure 14.9](#). The `AppDelegate` has been instantiated and is the `NSApplication`'s delegate, as you would expect. The Get Product Information button is linked to the `AppDelegate`'s `-getProductInformation:` action method. The `AppDelegate` also contains outlets for the appropriate items in the main window.

Figure 14.9. The MyXMLRPCClient `MainMenu.nib` file.



The Source Code

The interface of `AppController` is shown in [Listing 14.11](#).

Listing 14.11 `AppController` Interface in `AppController.h`

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

@interface AppController : NSObject
{
    IBOutlet NSPopUpButton *m_product;
    IBOutlet NSPopUpButton *m_color;
    IBOutlet NSTextField *m_price;
    IBOutlet NSTextField *m_stock;
    IBOutlet NSTextField *m_result;
}
- (IBAction) getProductInformation: (id) sender;
@end
```

The `AppController` manages the outlets of our user interface for convenience. One action is also connected to the Get Product Information button.

[Listing 14.12](#) contains the `-getProductInformation:` method that is called when the Get Product Information button is clicked. The user has already made her choice of product and color. This method alone is the equivalent of our PHP version of the client.

Listing 14.12 `-getProductInformation:` in `AppController.m`

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

```
{
```

```
WSMethodInvocationRef rpcCall;  
NSURL *rpcURL;  
NSString *methodName;  
NSMutableDictionary *params;  
NSArray *paramsOrder;  
NSDictionary *result;  
NSString *selectedProduct;  
NSString *selectedColor;  
  
//  
// 1. Define the location of the RPC service and method  
//  
  
// Create the URL to the RPC service  
rpcURL = [NSURL URLWithString:  
    @"http://www.triplesoft.com/xmlrpc/product_server.php"];  
  
// Assign the method name to call on the RPC service  
methodName = @"example.getProductInfo";  
  
// Create a method invocation  
// First parameter is the URL to the RPC service  
// Second parameter is the name of the RPC method to call  
// Third parameter is a constant to specify the XML-RPC protocol  
rpcCall = WSMethodInvocationCreate((CFURLRef)rpcURL,  
    (CFStringRef)methodName, kWSXMLRPCProtocol);  
  
//  
// 2. Set up the parameters to be passed to the RPC method  
//  
  
// Get the users choices  
selectedProduct = [m_product titleOfSelectedItem];  
selectedColor = [m_color titleOfSelectedItem];  
  
// Add the users choices to the dictionary to be passed as parameters  
params = [NSMutableDictionary dictionaryWithCapacity:2];  
[params setObject:selectedProduct forKey:selectedProduct];  
[params setObject:selectedColor forKey:selectedColor];  
  
// Create the array to specify the order of the parameter values  
paramsOrder = [NSArray arrayWithObjects:selectedProduct,  
    selectedColor, nil];  
  
// Set the method invocation parameters  
// First parameter is the method invocation created above  
// Second parameter is a dictionary containing the parameters themselves  
// Third parameter is an array specifying the order of the parameters  
WSMethodInvocationSetParameters(rpcCall, (CFDictionaryRef)params,  
    (CFArrayRef)paramsOrder);  
  
//  
// 3. Make the call and parse the results!  
//
```

```

// Invoke the method which returns a dictionary of results
result = (NSDictionary*)WSMethodInvocationInvoke(rpcCall);

// If the results are a fault, display an error to the user with the
// fault code and descriptive string
if (WSMethodResultIsFault((CFDictionaryRef)result)) {
    NSRunAlertPanel ([NSString stringWithFormat:@"Error %@",
        [result objectForKey: (NSString*)kWSFaultCode]],
        [result objectForKey: (NSString*)kWSFaultString],
        @"OK", @"", @"");
} else {
    // Otherwise, pull the results from the dictionary as an array
    NSArray *array = [result objectForKey:
        (NSString*)kWSMethodInvocationResult];

    // Display the entire array result from the server
    [m_result setStringValue: [array description]];

    // Display the specific fields we are interested in (price, stock)
    [m_price setStringValue: [array objectAtIndex: 2]];
    [m_stock setStringValue: [array objectAtIndex: 3]];
}

// Release those items that need to be released
[params release];
params = nil;
[paramsOrder release];
paramsOrder = nil;
[result release];
result = nil;
}

```

Let's look at the method in stages. The first thing we do in [Listing 14.13](#) is to define all the variables we will be using in the method. A lot are listed here, but most are straightforward: `NSArray`s, `NSString`s, and `NSDictionary`s (mutable and otherwise). However, one item is new. The `WSMethodInvocationRef` variable is used to track and manage the object that handles our XML-RPC call. You'll see how this works shortly.

Listing 14.13 - `getProductInformation`: —Part 1 in `AppController.m`

```

- (IBAction) getProductInformation: (id) sender
{
    WSMethodInvocationRef rpcCall;
    NSURL                *rpcURL;
    NSString              *methodName;
    NSMutableDictionary *params;
    NSArray               *paramsOrder;
    NSDictionary          *result;
    NSString              *selectedProduct;
    NSString              *selectedColor;

    //
    // 1. Define the location of the RPC service and method
    //

```

```

// Create the URL to the RPC service
rpcURL = [NSURL URLWithString:
    @"http://www.triplesoft.com/xmlrpc/product_server.php"];

// Assign the method name to call on the RPC service
methodName = @"example.getProductInfo";

// Create a method invocation
// First parameter is the URL to the RPC service
// Second parameter is the name of the RPC method to call
// Third parameter is a constant to specify the XML-RPC protocol
rpcCall = WSMethodInvocationCreate((CFURLRef)rpcURL,
    (CFStringRef)methodName, kWSXMLRPCProtocol);

```

Next, we want to create an `NSURL` that identifies the server that we will be calling. We also want to assign the `methodName` variable to contain the name of the RPC method. Last, we create the `WSMethodInvocation` by calling `WSMethodInvocationCreate`, part of the Web Services functionality. This method takes three parameters. The first is the URL to the RPC service. The second is the name of the RPC method. The third is a constant to specify which protocol to use—in our case, `kWSXMLRPCProtocol`. Other protocol options include SOAP. These are all defined in `WSMethodInvocation.h`.

In [Listing 14.14](#), we set up the parameters to be passed to the RPC based on the users choices. First, we obtain the values of the pop-up buttons for product and color as strings. We then create an `NSMutableDictionary` capable of storing two items using the `+dictionaryWithCapacity:` class method. Then, using the `-setObject:forKey:` method of `NSMutableDictionary`, we add the strings to the dictionary.

Listing 14.14 - `getProductInformation:` —Part 2 in `AppController.m`

```

//
//      2. Set up the parameters to be passed to the RPC method
//

// Get the users choices
selectedProduct = [m_product titleOfSelectedItem];
selectedColor = [m_color titleOfSelectedItem];

// Add the users choices to the dictionary to be passed as parameters
params = [NSMutableDictionary dictionaryWithCapacity:2];
[params setObject:selectedProduct forKey:selectedProduct];
[params setObject:selectedColor forKey:selectedColor];

// Create the array to specify the order of the parameter values
paramsOrder = [NSArray arrayWithObjects:selectedProduct, selectedColor, nil];

// Set the method invocation parameters
// First parameter is the method invocation created above
// Second parameter is a dictionary containing the parameters themselves
// Third parameter is an array specifying the order of the parameters
WSMethodInvocationSetParameters(rpcCall, (CFDictionaryRef)params,
    (CFArrayRef)paramsOrder);

```

Note

Another way to have done this would be to use an `NSDictionary` as opposed to an `NSMutableDictionary`. I chose to use an `NSMutableDictionary` in case I wanted to add more parameters later on—one less thing to change. Using an `NSDictionary`, you might consider the following:

```
NSDictionary *d = [NSDictionary dictionaryWithObjectsAndKeys:
    selectedProduct, selectedProduct,
    selectedColor, selectedColor,
    nil];
```

Remember that XML-RPC requires parameters to be in the correct order; therefore, we must create an `NSArray` to specify that order. We simply add the items in the proper order using the `NSArray +arrayWithObject:` class method. Failure to include this parameter will cause your server to become confused because the parameter order is unspecified.

We use the `WSMethodInvocationSetParameters` function to attach the parameters to the `WSMethodInvocationRef`. This function takes three parameters. The first is the reference to the method invocation created earlier. The second is the dictionary containing the parameters themselves. The third is the array specifying the order of the parameters.

In [Listing 14.15](#), we make the call to the RPC by calling the `WSMethodInvocationInvoke` function and passing the `WSMethodInvocationRef`. This is equivalent to the `$client->send($message)` method in the PHP client version. This single function handles locating the remote server, transmitting the message to the client via the HTTP protocol, and accepting any resulting data from the server as an `NSDictionary`.

Listing 14.15 - `getProductInformation`: —Part 3 in `AppController.m`

```
//
// 3. Make the call and parse the results!
//

// Invoke the method which returns a dictionary of results
result = (NSDictionary*)WSMethodInvocationInvoke(rpcCall);

// If the results are a fault, display an error to the user with the
// fault code and descriptive string
if (WSMethodResultIsFault((CFDictionaryRef)result)) {
    NSRunAlertPanel([NSString stringWithFormat:@"Error %@",
        [result objectForKey:(NSString*)kWSFaultCode]],
        [result objectForKey:(NSString*)kWSFaultString], @"OK", @"", @"");
} else {
    // Otherwise, pull the results from the dictionary as an array
    NSArray *array = [result objectForKey:
        (NSString*)kWSMethodInvocationResult];

    // Display the entire array result from the server
    [m_result setValue:[array description]];

    // Display the specific fields we are interested in (price, stock)
```

```

    [m_price setStringValue: [array objectAtIndex: 2]];
    [m_stock setStringValue: [array objectAtIndex: 3]];
}

// Release those items that need to be released
[params release];
params = nil;
[paramsOrder release];
paramsOrder = nil;
[result release];
result = nil;
}

```

We then check the results for an error by calling the `WSMethodResultIsFault` function, passing the `NSDictionary` typecast as a `CFDictionaryRef`—[toll free bridging between CoreFoundation and Foundation](#) makes this simple. If the function returns `TRUE`, we pull `kWSFaultCode` and `kWSFaultString` from the result and display them to the user. The `kWSFaultString` is the error returned from the server in the `Serr` variable.

Assuming that there are no errors, we pull the array of results from the dictionary as `kWSMethodInvocationResult`. When we have the array, we know what it should contain and in what order. The first thing we do is display the entire array (for debugging purposes) by calling the `-description` method of `NSArray`. This allows us to quickly see all fields in the result array and display them to the user. Next, we pull just the price and stock information as array objects `2` and `3` to display to the user as well. Remember that array objects `0` and `1` are an echo of the product name and color that we passed to the server.

At the end of the line, we merely clean up after ourselves and release all the memory that needs to be released.

So, what do you think? It's not so hard to harness the power of distributed computing across languages and platforms!

[\[Team LiB \]](#)

◀ PREVIOUS NEXT ▶

Try This

A client/server architecture is a great place to experiment with different implementations. Here are some ideas to keep you busy with this one.

Collect the list of product names and colors used in the HTML form in the PHP client from the server itself. Assume that the client knows nothing about what products are available. The server should implement functions named `getProductNames` and `getProductColors` to be listed to the user. Once you get it working in the PHP client, implement it in the Cocoa client as well.

Use MySQL (or another database) as your data source in the server. If you don't want to explore MySQL (or another database), check out the Flat File DataBase (FFDB) PHP Library by John Papandriopoulos. John's library offers a simple way to use database functionality quickly and easily on your server using PHP:

<http://ffdb-php.sourceforge.net/>

Conclusion

I hope that XML-RPC excites you as much as it does me. I find PHP and Web-based implementations to be an incredible way to open up your creativity to users of all platforms via a Web browser. You can implement so many excellent technologies using these methods. Explore the possibilities of XML-RPC and create something new for the world to explore!

Chapter 15. SOAP

"Scrubba Dub Dub, PHP and Cocoa in the tub."

—I'll admit it; I said this one.

A note to those of you reading this book in order: This chapter is very similar to [Chapter 14](#), "XML-RPC," with differences only where the XML-RPC and SOAP examples vary. The goal of these two chapters is to show the same project implemented in both XML-RPC and SOAP and to have them both stand alone should someone read them separately. If you already understand the concepts outlined in [Chapter 14](#), you can skip ahead and just look at the source code explanations in this chapter.

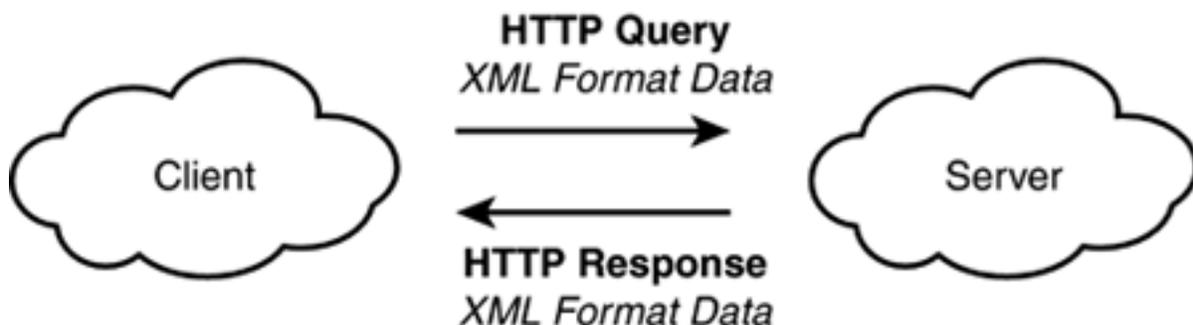
SOAP, the Simple Object Access Protocol, is a Web Service similar to XML-RPC in function but different in form. Like XML-RPC, SOAP allows you to transmit complex data structures to remote computers for processing and response and uses XML to format its data. If you have a need for a reliable client/server architecture reaching across platforms, SOAP might be for you.

Although SOAP can, and often does, use the HTTP protocol as its transport mechanism, it need not be bound by this limitation. HTTP is, by far, the most widely implemented protocol across multiple platforms; however, SOAP allows you to use others as well, including SMTP. In this chapter, we will use HTTP as our example's transport mechanism.

SOAP can be implemented in any of a number of programming languages on just about every available platform. You can choose C/C++ (including Carbon), Java, Lisp, Objective-C (including Cocoa), Perl, PHP, Python, BASIC, AppleScript, and many more. This offers a lot of flexibility to the programmer looking for a robust solution with minimal limitations.

Both XML-RPC and SOAP are useful when you need to implement a data-based client/server architecture. Although XML-RPC and SOAP are "Web Services", normally they will not return fully formatted HTML, but instead just the raw data. Remember, you don't know what format your client might be—it might be a Web browser, it might be a native application with a Mac OS X or Windows GUI, or it might be a cell phone. Therefore, your server should return data only, leaving the formatting of the data to the client. This is a perfect use for XML. [Figure 15.1](#) shows this concept.

Figure 15.1. A client and server via SOAP.



The official SOAP specification can be found at <http://www.w3.org/2000/xp/Group/>. It has extremely

detailed information on the protocol that we will use in this chapter.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

The Result

The project in this chapter is the same project we used in [Chapter 14](#). One thing I found while learning about Web Services is that it was very difficult to find concrete examples that accepted multiple parameters in and delivered multiple parameters out—like the real-world. By showing a single example using XML-RPC and SOAP as well as both PHP and Objective-C, I hope this will take some of the mystery out of these exciting protocols.

This project shows how to create a SOAP server written in PHP and running under the Apache Web server. We then create a PHP client as well as Cocoa Objective-C client. This shows that one server can handle multiple clients using multiple implementation mechanisms. Although the Cocoa client will only run under Mac OS X, the PHP client can be used on any computer with a Web browser.

Our example is a product information system. We sell many different styles of bags in a rainbow of colors—everything from book bags to handbags to grocery bags and more. Using one of our clients, the user simply picks a type of bag and a color, and the server returns the price and number of bags currently in stock that matches those criteria. For the purposes of this demo, any bag in the color brown is out of stock, whereas all other colors return a random stock value. Normally, you would pull this information from a database, but this is left as an exercise to the reader.

The PHP client is served via Apache and is essentially a Web form. The client accepts data from the user, sends the request to the server, formats the resulting data appropriately, and displays it in a Web browser.

[Figure 15.2](#) shows the PHP client running in Internet Explorer under Mac OS X although it could run in any modern Web browser. The user is presented with a form that allows her to choose which type of bag and which color she wants price and stock information for. Once chosen, she clicks the Get Product Information button, which passes the form data on to the server.

Figure 15.2. The PHP SOAP client before submitting a query.



Once the server receives and processes the data, it returns the resulting data (or an error) to the client. The client is then responsible for displaying the results to the user. [Figure 15.3](#) shows our client displaying the original XML-format query—also called the request or message—the formatted results, and then the details of the response from the server that were parsed to produce the formatted results. Normally, you wouldn't display the XML-formatted information to the user, but it is very helpful to see the complete picture when you're learning something new. There is much more resulting data than can be shown here, which will be covered in detail later in this chapter.

Figure 15.3. The PHP SOAP client after submitting a query and receiving the results.



The Cocoa client, shown in [Figure 15.4](#), works similarly in that it collects data from the user and then contacts the server to perform the price and stock lookup. However, after it receives and parses the response, it displays it within its own window differently than the previous Web browser example. It still uses HTTP as the transport mechanism, but does not display the results using HTML.

Figure 15.4. The Cocoa SOAP client after submitting a query and receiving the results.



Let's look at how this is all put together!

[\[Team LiB \]](#)

The PHP Server

First, we will look at the server side. As mentioned, the server is written in PHP, but it can be written in almost any language. We chose PHP because it is reliable, straightforward, and powerful. Our `product_server.php` file is served by the Apache Web server running under Mac OS X just as any standard PHP file would be. The client accesses it via a standard HTTP URL. In this case, we have placed the file at `http://www.triplesoft.com/soap/product_server.php`. However, typing that URL into your Web browser will simply produce an error because SOAP servers require specifically formatted information passed by the client. [Figure 15.5](#) shows the fault information returned by the server.

Figure 15.5. The PHP SOAP server being called directly from the Web browser.



An important thing to note is that both the PHP server and client require the use of a third-party PHP library by Dietrich Ayala of NuSphere Corporation. The `NuSOAP` library and detailed information on its use can be found at <http://dietrich.ganx4.com/nusoap/>. This library contains one PHP file, `nusoap.php`, which is easily included in your PHP source. The distribution also contains documentation as well as licensing information.

Let's look at the complete PHP source code of the server and then I will explain it in sections.

[Listing 15.1](#) contains the complete PHP SOAP server code. This simple server offers a concrete example of

the protocol accepting multiple parameters and returning multiple parameters as an array, which is most likely what you will need in a real-life scenario.

Listing 15.1 product_server.php

```
<?php
include("nusoap.php");

// Array of products and prices, in real-life you might use a SQL database
$products = array (
    "Grocery Bag" => "9.00",
    "Book Bag" => "18.00",
    "Hand Bag" => "32.00",
    "Garment Bag" => "48.00",
    "Old Bag" => "0.31"
);

// Create a new SOAP server instance
$server = new soap_server;

// Enable debugging, to be echoed via client
$server->debug_flag = true;

// Register the get_product_info function for publication
$server->register("get_product_info");

// Begin the HTTP listener service and exit
$server->service($HTTP_RAW_POST_DATA);

// Our published service
function get_product_info($product_name, $product_color)
{
    global $products;
    $err = "";

    // Verify values
    if (isset($product_name) && isset($product_color)) {

        // Attempt to find the product and price
        foreach ($products as $key => $element) {
            if ($key == $product_name) {
                $product_price = $element;
            }
        }

        // For this demo, all Browns are out of stock and all other colors
        // produce a random in-stock quantity - probably wouldn't work like
        // this in real-life, you would probably use a SQL database instead
        if ($product_color == "Brown") {
            $product_stock = 0;
        } else {
            $product_stock = rand(1, 100);
        }

        // If not found, report error
```

```

        if (!isset($product_price)) {
            $err = "No product named '" . $product_name . "' found.";
        }
    } else {
        $err = "Two string parameters (product name and color) are required.";
    }

    // If an error was generated, return it, otherwise return the proper data
    if ($err) {
        return new soap_fault("Client", "product_server.php", $err, $err);
    } else {
        // Create an array and fill it with the data to return

        $result_array = array("product_name" => $product_name,
                               "product_color" => $product_color,
                               "product_price" => $product_price,
                               "product_stock" => $product_stock);

        return $result_array;
    }
}

exit();

?>

```

Let's look at the server code by section. First, in [listing 15.2](#), you will notice the include statement for the `nusoap.php` file. This file is required by both clients and servers and provides much of the functionality to interface PHP with the underlying SOAP protocol. Using NuSOAP makes life as a SOAP developer much easier.

Listing 15.2 Initial Steps in `product_server.php`

```

<?php
include("nusoap.php");

// Array of products and prices, in real-life you might use a SQL database
$products = array (
    "Grocery Bag" => "9.00",
    "Book Bag" => "18.00",
    "Hand Bag" => "32.00",
    "Garment Bag" => "48.00",
    "Old Bag" => "0.31"
);

// Create a new SOAP server instance
$server = new soap_server;

// Enable debugging, to be echoed via client
$server->debug_flag = true;

// Register the get_product_info function for publication

```

```
$server->register("get_product_info");
```

```
// Begin the HTTP listener service and exit
```

```
$server->service($HTTP_RAW_POST_DATA);
```

Next, we create an array of products with prices, appropriately named `$products`. In a real-life scenario, you will most likely use some sort of database for this information. MySQL is a common choice when using PHP and Apache, but other options exist as well.

We now create the SOAP server instance using the PHP `new` function to create a `soap_server` object. This functionality is provided by NuSOAP. Once the server is created, we use the `$server` variable to manage it. For debugging purposes, we set the `debug_flag` instance variable of the server object to `true`. This will provide a wealth of debug information in our Web browser when we access the server, and it is highly recommended during development.

After the server has been created, we need to `register` the functions that our server implements for publication. In this example, we only publish one function, `get_product_info`. At this stage, you need only register the name; the expected parameters to this function are handled automatically by NuSOAP and SOAP.

Last, we need to begin the process of listening to HTTP. By calling the `service` method of the server object, we begin the process of listening for requests for our published service. In reality, when the server is executed by means of a client, you are pretty much guaranteed that a request is forthcoming, so we won't be waiting for long.

The `get_product_info` function in [Listing 15.3](#) is the main and only published service of this server. Although servers can publish multiple services, we only implement one in this example. This function does all the work—verifies the incoming parameters, performs any calculations or database lookups, and returns the results to the client. This function is automatically called when the server receives the request based on information within the XML-formatted data sent to the server. [Listing 15.4](#) shows an example of the XML sent to the server that triggers this function call.

Listing 15.3 The `get_product_info` Method in `product_server.php`

```
// Our published service
function get_product_info($product_name, $product_color)
{
    global $products;
    $err = "";

    // Verify values
    if (isset($product_name) && isset($product_color)) {

        // Attempt to find the product and price
        foreach ($products as $key => $element) {
            if ($key == $product_name) {
                $product_price = $element;
            }
        }

        // For this demo, all Browns are out of stock and all other colors
        // produce a random in-stock quantity - probably wouldn't work like
```

```

// this in real-life, you would probably use a SQL database instead
if ($product_color == "Brown") {
    $product_stock = 0;
} else {
    $product_stock = rand(1, 100);
}

// If not found, report error
if (!isset($product_price)) {
    $err = "No product named '" . $product_name . "' found.";
}

} else {
    $err = "Two string parameters (product name and color) are required.";
}

// If an error was generated, return it, otherwise return the proper data
if ($err) {
    return new soap_fault("Client", "product_server.php", $err, $err);
} else {
    // Create an array and fill it with the data to return

    $result_array = array("product_name" => $product_name,
        "product_color" => $product_color,
        "product_price" => $product_price,
        "product_stock" => $product_stock);

    return $result_array;
}
}

```

Listing 15.4 The XML-Formatted Data Sent to `product_server.php`

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:si="http://soapinterop.org/xsd">
  <SOAP-ENV:Body>
    <ns1:get_product_info xmlns:ns1="http://testuri.org">
      <product_name xsi:type="xsd:string">Book Bag</product_name>
      <product_color xsi:type="xsd:string">Black</product_color>
    </ns1:get_product_info>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Looking at [Listing 15.3](#) again, we first allow access to the global `$products` array we defined previously. `$err` will contain any error code that is generated during the execution of the server, for now we initialize it to "".

Next, we verify that the expected variables are set. We explicitly expect the `$product_name` and `$product_color` parameters to be passed in from the client as shown in the body of the SOAP message in [Listing 15.4](#). NuSOAP conveniently handles extracting the parameters from the SOAP XML message and passing them to our PHP function. If either of these parameters is not sent, it is considered an error and the user is alerted.

When the parameter values have parsed properly, we can then begin to use the data they contain. The first thing to do is peruse the `$products` array using `foreach` for the chosen product. Because we limit the users choices in the client, we should always get a match here. Once found, we set the `$product_price` variable accordingly. If the `$product_price` is not found, which should never happen in our simple example, we return a human-readable error in `$err`, which is shown a few lines further down.

Next, we figure out the number of the product in the chosen color that is in stock. For the purposes of this demo, all products chosen in brown are out of stock. All other colors produce a random stock value. In reality, you might pull this information from a SQL database using MySQL.

Last, if an error was returned, `$err` is set; we return the error by creating and returning a `soap_fault` object. The `soap_fault` object accepts up to four parameters. The first, `faultcode`, is either 'client' or 'server' and tells where the fault of the error occurred. In this case, the error is always considered one made by the client, such as passing the wrong parameters. If there were a specific server error, such as a database lookup failure, you would pass 'server' for this parameter. The second, `faultactor`, is the URL of the script that found the error. It can be helpful to fill this in when multiple actors (or servers) are involved in a process. The third, `faultstring`, is the human-readable error string and the fourth, `faultdetail`, can be an error code, and so on. You can see an example of an error returned in [Figure 15.5](#).

However, if all is well, we create and return our result array. Remember, our server returns an array of data to the client. We do this by simply creating a standard PHP array and adding the items we want to return. The product name and color are the same items passed to the server—we return them for convenience to the client because HTTP is a stateless protocol. The product price and stock are calculated by our server. [Listing 15.5](#) shows an example of the data returned from the server—note the parameters are listed by name.

Listing 15.5 The XML-Formatted Data Returned from `product_server.php`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:si="http://soapinterop.org/xsd">
  <SOAP-ENV:Body>
    <get_product_infoResponse>
      <soapVal>
        <product_name xsi:type="xsd:string">Book Bag</product_name>
        <product_color xsi:type="xsd:string">Black</product_color>
        <product_price xsi:type="xsd:string">18.00</product_price>
        <product_stock xsi:type="xsd:int">51</product_stock>
      </soapVal>
    </get_product_infoResponse>
  </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

The very last thing to do is `exit` and end the PHP code. [Listing 15.6](#) performs this task.

Listing 15.6 Last but not Least in `product_server.php`

```
exit();
```

```
?>
```

That is all there is to the server. Now let's look at the PHP client.

[\[Team Lib \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

The PHP Client

Our first client is also written in PHP, just as the server. As you've seen, however, it could be written in just about any language. Later in this chapter, we will rewrite the client using Objective-C and Cocoa. The `product_client.php` file is served by the Apache Web server running under Mac OS X just as any standard PHP file would be. The client file is what the user will access via her Web browser. In this case, we have placed the file at `http://www.triplesoft.com/soap/product_client.php`. Typing that URL in your Web browser will produce the Web page shown in [Figure 15.2](#).

When the user clicks the Get Product Information button in the Web form, the pop-up menu selections are sent to the server as XML-formatted data. The server then processes the selections and returns the price and stock information as previously discussed. [Figure 15.3](#) shows what the users Web browser looks like after the server responds.

[Listing 15.7](#) contains the complete PHP SOAP client code. This simple client offers a concrete example of the protocol passing multiple parameters and parsing multiple parameters in return.

Listing 15.7 `product_client.php`

```
<html >
<head><title>PHP SOAP Exampl e</title></head>
<body>
<basefont size="2" face="Verdana, Ari al ">

<?php
include("nusoap. php");

// Show the results of the previous query
// if the parameter there are results to show
if (($HTTP_POST_VARS["product_name"] != "") &&
    ( $HTTP_POST_VARS["product_col or"] != "")) {

    // Create the parameters array
    $params = array("product_name" => $HTTP_POST_VARS["product_name"],
        "product_col or" => $HTTP_POST_VARS["product_col or"]);

    // Create a new soap client endpoint specifying the server location
    $client = new soapclient("http://www.triplesoft.com/soap/product_server. php");

    // Enable debugging
    $client->debug_flag = true;

    // Call the function, passing the parameter array
    $response = $client->call("get_product_i nfo", $params);

    // Echo request detail
    echo "<pre><b>Request Detail:<br /></b><xmp>". $client->request. "</xmp></pre>";

    // If no response was returned there was a fatal
    // error (no network conn, no server, etc.)
```

```

// You might prefer to use the $client->getError()
// method for more detailed information.
if (!$response) { die("Send failed."); }

// If no error, display the results
if (!$client->fault) {

    // Extract the values from the response
    if (count($response) == 4) {

        foreach ($response as $key => $element) {
            $$key = $element;
        }

        print "<br />";
        print "<br />Product: <b>" . $product_name . "</b>";
        print "<br />Color: <b>" . $product_color . "</b>";
        print "<br />Price: <b>$" . $product_price . "</b>";
        print "<br />In-Stock: <b>" . $product_stock . "</b>";
        print "<br />";

    } else {

        print "<br />";
        print "<br />Incorrect number of items in array.
            Should be 4 but there are <b>"
            . count($response) . "</b>.";
        print "<br />";

    }

    // Echo response detail
    echo "<pre><b>Response Detail:<br /></b><xmp>".
        $client->response. "</xmp></pre>";

} else {
    print "<br />";
    print "<br /><b>Fault Code:</b> " . $client->faultcode;
    print "<br /><b>Fault Actor:</b> " . $client->faultactor;
    print "<br /><b>Fault String:</b> " . $client->faultstring;
    print "<br /><b>Fault Detail:</b> " . $client->faultdetail; // serialized?
    print "<br />";
}

// Echo debug log at the bottom since it can be large
echo "<pre><b>Debug log:<br /></b>". $client->debug_str. "</pre>";

print "<hr>";

}
?>

```

```

<form action="<?php echo "$PHP_SELF";?>" method="POST">
    <select name="product_name">
        <option label="Book Bag" value="Book Bag" selected>Book Bag</option>
        <option label="Garment Bag" value="Garment Bag">Garment Bag</option>
        <option label="Grocery Bag" value="Grocery Bag">Grocery Bag</option>
    </select>

```

```

        <opti on l abel ="Hand Bag" val ue="Hand Bag">Hand Bag</opti on>
        <opti on l abel ="Ol d Bag" val ue="Ol d Bag">Ol d Bag</opti on>
</select>
in
<sel ect name="product_col or">
    <opti on l abel ="Bl ack" val ue="Bl ack" sel ected>Bl ack</opti on>
    <opti on l abel ="Bl ue" val ue="Bl ue">Bl ue</opti on>
    <opti on l abel ="Brown" val ue="Brown">Brown</opti on>
    <opti on l abel ="Green" val ue="Green">Green</opti on>
    <opti on l abel ="Red" val ue="Red">Red</opti on>
    <opti on l abel ="Whi te" val ue="Whi te">Whi te</opti on>
</select>
<butt on type="submi t" name="Get Product Informati on"
    val ue="Get Product Informati on">Get Product Informati on</butt on>
</form>

</body>
</html >

```

Let's look at the client code by section. Unlike our server PHP code, our client PHP code contains both HTML and PHP intermixed. The first section, shown in [Listing 15.8](#), does little more than implement the standard HTML tags required for a Web page.

Listing 15.8 In the Beginning of `product_client.php`

```

<html >
<head><titl e>PHP SOAP Exampl e</titl e></head>
<body>
<basefont size="2" face="Verdana, Ari al ">

```

[Listing 15.9](#), however, gets us into the code that handles the SOAP fat of our client.

Listing 15.9 In the Middle of `product_client.php`

```

<?php
incl ude("nusoap. php");

// Show the results of the previous query
// if the parameter there are results to show
if (($HTTP_POST_VARS["product_name"] != "") &&
    ($HTTP_POST_VARS["product_col or"] != "")) {

    // Create the parameters array
    $params = array("product_name" => $HTTP_POST_VARS["product_name"],
        "product_col or" => $HTTP_POST_VARS["product_col or"]);

    // Create a new soap client endpoint specifying the server location
    $client =
        new soapclient("http://www.tripl esoft. com/soap/product_server. php");

    // Enable debuggi ng
    $client->debug_flag = true;

```

```

// Call the function, passing the parameter array
$response = $client->call("get_product_info", $params);

// Echo request detail
echo "<pre><b>Request Detail:<br /></b><xmp>". $client->request. "</xmp></pre>";

// If no response was returned there was a
// fatal error (no network conn, no server, etc.)
// You might prefer to use the $client->getError()
// method for more detailed information.
if (!$response) { die("Send failed."); }

// If no error, display the results
if (!$client->fault) {

    // Extract the values from the response
    if (count($response) == 4) {

        foreach ($response as $key => $element) {
            $$key = $element;
        }

        print "<br />";
        print "<br />Product: <b>" . $product_name . "</b>";
        print "<br />Color: <b>" . $product_color . "</b>";
        print "<br />Price: <b>$" . $product_price . "</b>";
        print "<br />In-Stock: <b>" . $product_stock . "</b>";
        print "<br />";

    } else {

        print "<br />";
        print "<br />Incorrect number of items in array.
            Should be 4 but there are <b>"
            . count($response) . "</b>.";
        print "<br />";

    }

    // Echo response detail
    echo "<pre><b>Response Detail:<br /></b><xmp>".
        $client->response. "</xmp></pre>";

} else {
    print "<br />";
    print "<br /><b>Fault Code:</b> " . $client->faultcode;
    print "<br /><b>Fault Actor:</b> " . $client->faultactor;
    print "<br /><b>Fault String:</b> " . $client->faultstring;
    print "<br /><b>Fault Detail:</b> " . $client->faultdetail; // serialized?
    print "<br />";
}

// Echo debug log at the bottom since it can be large
echo "<pre><b>Debug log:<br /></b>". $client->debug_str. "</pre>";

print "<hr>";

```

```
}  
?>
```

[Listing 15.9](#) begins by including the `nusoap.php` file. We then verify that we have values for both of the `$HTTP_POST_VARS` that we are interested in. If both `product_name` and `product_color` are set, we continue creating the SOAP message by first adding them to the `$params` array by name.

We then create the `soapclient` object using PHP `new` function. The `soapclient` takes one parameter, the URL to the server—in our case, our PHP server script living at `http://www.tripl esoft.com/soap/product_server.php`. It returns a client object used to send the message to the server as the variable `$client`.

At this point, you can enable debug mode by setting the `debug_flag` variable of the `$client` object to `true`. In either case, we send the message using the `send` method of the client object. This method takes two parameters, the name of the published service, `get_product_info`, and the parameters to pass to it. It single-handedly locates the remote server, transmits the message to the client via the chosen protocol, and accepts any resulting data from the server as the `$response` variable. For debugging purposes, we then echo the request detail by calling the client's `request` method. If there is no response, the function dies with an error message; otherwise, it continues parsing the result.

If the client object's `fault` method returns an empty result, there is no error and we can continue with the parsing. Otherwise, we display the `faultcode`, `faultactor`, `faultstring`, and `faultdetail` results to the user as the error. As you recall, these are sent from the server.

Assuming that there is no error, we verify the size of the array is four; otherwise, there is an error, and we display it accordingly. Remember, the server returns four values in the array of results. We then use `foreach` to extract the keys and elements in the array for display back to the user. For debugging purposes, we then echo the response detail by calling the client's `response` method. We also show the debug log by calling the client's `debug_str` method.

That ends the PHP portion of our file.

[Listing 15.10](#) is nothing more than the HTML form portion of our client file. This portion appears each time the page is displayed as seen in [Figure 15.2](#). It simply contains the pop-up menus and Get Product Information button and calls back to itself as the form action. Although this example hard-codes the product names and colors into the form, a real-life application might query the server for the list of products and list of available colors. See the "[Try This](#)" section of this chapter for your assignment to do just that!

Listing 15.10 In the End of `product_client.php`

```
<form action="<?php echo "$PHP_SELF"; ?>" method="POST">  
  <select name="product_name">  
    <option label="Book Bag" value="Book Bag" selected>Book Bag</option>  
    <option label="Garment Bag" value="Garment Bag">Garment Bag</option>  
    <option label="Grocery Bag" value="Grocery Bag">Grocery Bag</option>  
    <option label="Hand Bag" value="Hand Bag">Hand Bag</option>  
    <option label="Old Bag" value="Old Bag">Old Bag</option>  
  </select>  
  in  
  <select name="product_color">
```

```
<opti on l abel ="Bl ack" val ue="Bl ack" sel ected>Bl ack</opti on>
<opti on l abel ="Bl ue" val ue="Bl ue">Bl ue</opti on>
<opti on l abel ="Brown" val ue="Brown">Brown</opti on>
<opti on l abel ="Green" val ue="Green">Green</opti on>
<opti on l abel ="Red" val ue="Red">Red</opti on>
<opti on l abel ="Whi te" val ue="Whi te">Whi te</opti on>
</sel ect>
<butt on type="submi t" name="Get Product Informati on"
    val ue="Get Product Informati on">Get Product Informati on</butt on>
</form>

</body>
</html >
```

Now let's see how we can implement this client using Objective-C.

[\[Team LiB \]](#)

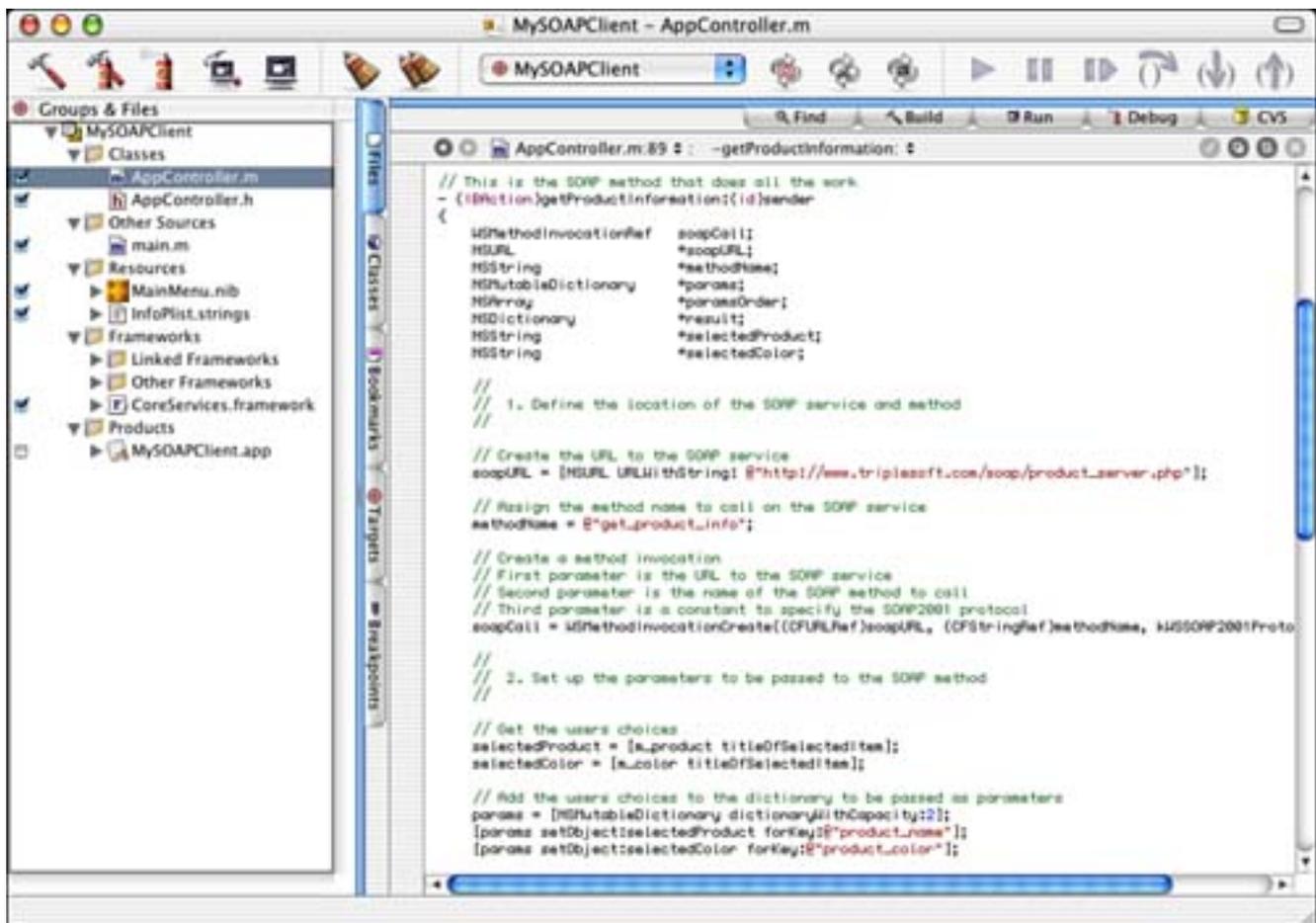
◀ PREVIOUS

NEXT ▶

The Cocoa Client

Figure 15.6 shows the MySOAPClient project in Project Builder. The project is a simple Cocoa application. We use an `AppController` to implement the `-applicationShouldTerminateAfterLastWindowClosed:` delegate method. For the sake of convenience, we also implement the `-getProductInformation:` method in the `AppController` and link it to the Get Product Information button in our main application window shown in Figure 15.4.

Figure 15.6. The MySOAPClient project.

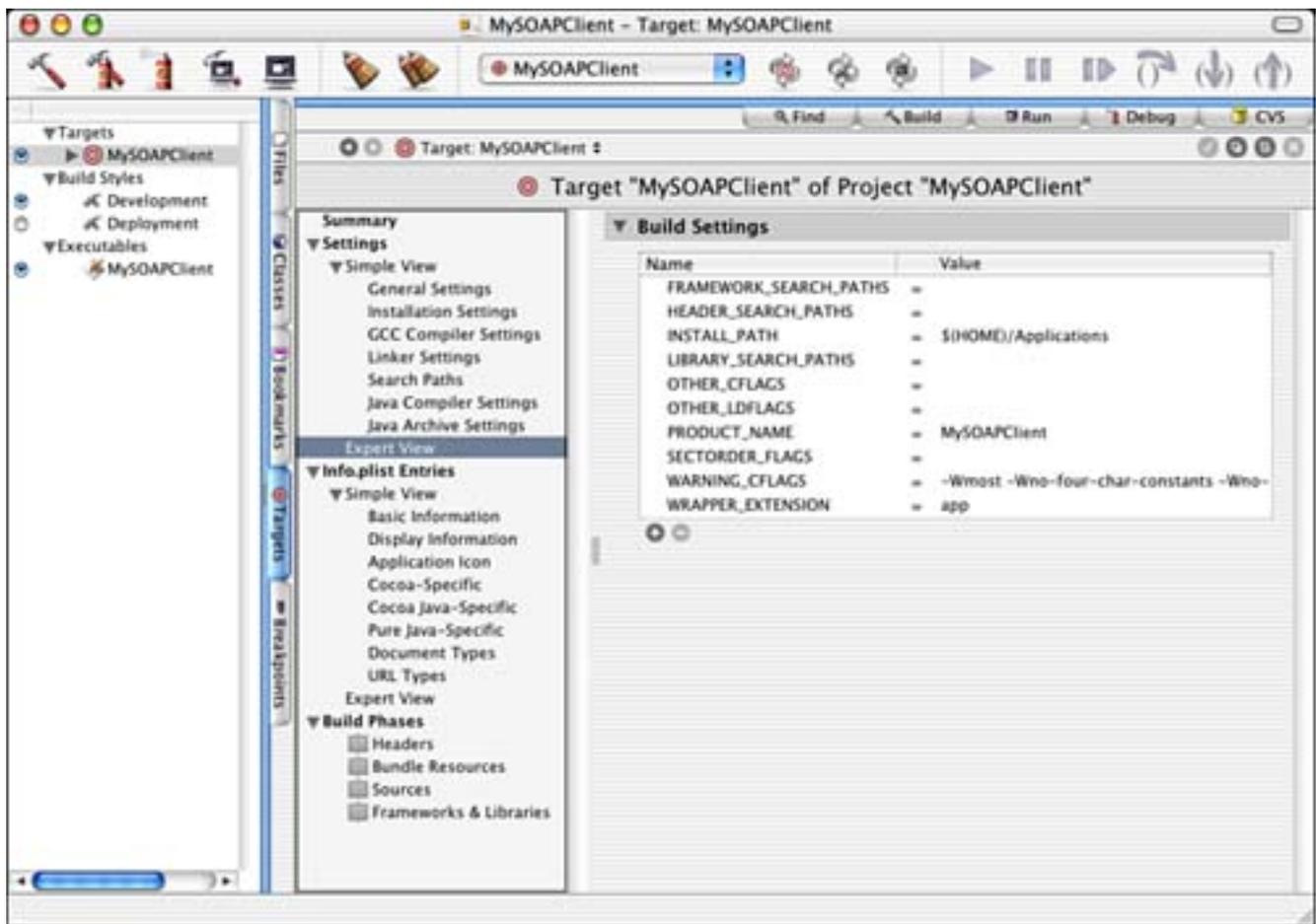


One thing to note is the addition of the `CoreServices.framework` in the project. This framework contains functionality that allows us to access Web Services such as XML-RPC and SOAP. You'll see more of what this brings us when we look at the source code later.

The Project Settings

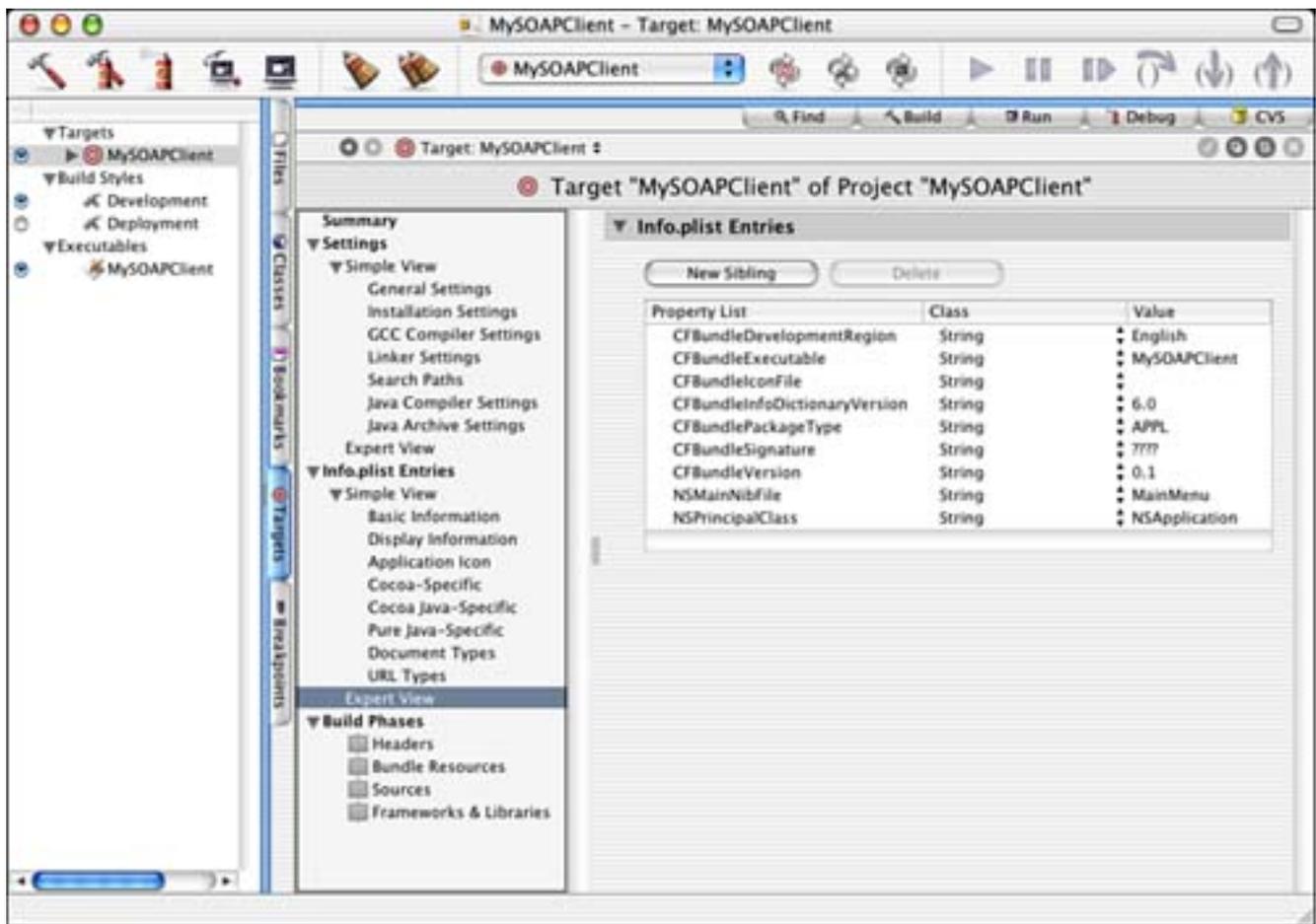
The Target settings in Figure 15.7 are straightforward for a simple Cocoa application.

Figure 15.7. The MySOAPClient Target settings.



The InfoPlist entries in [Figure 15.8](#) are also straightforward. Because this is a simple Cocoa application, there are no surprises here.

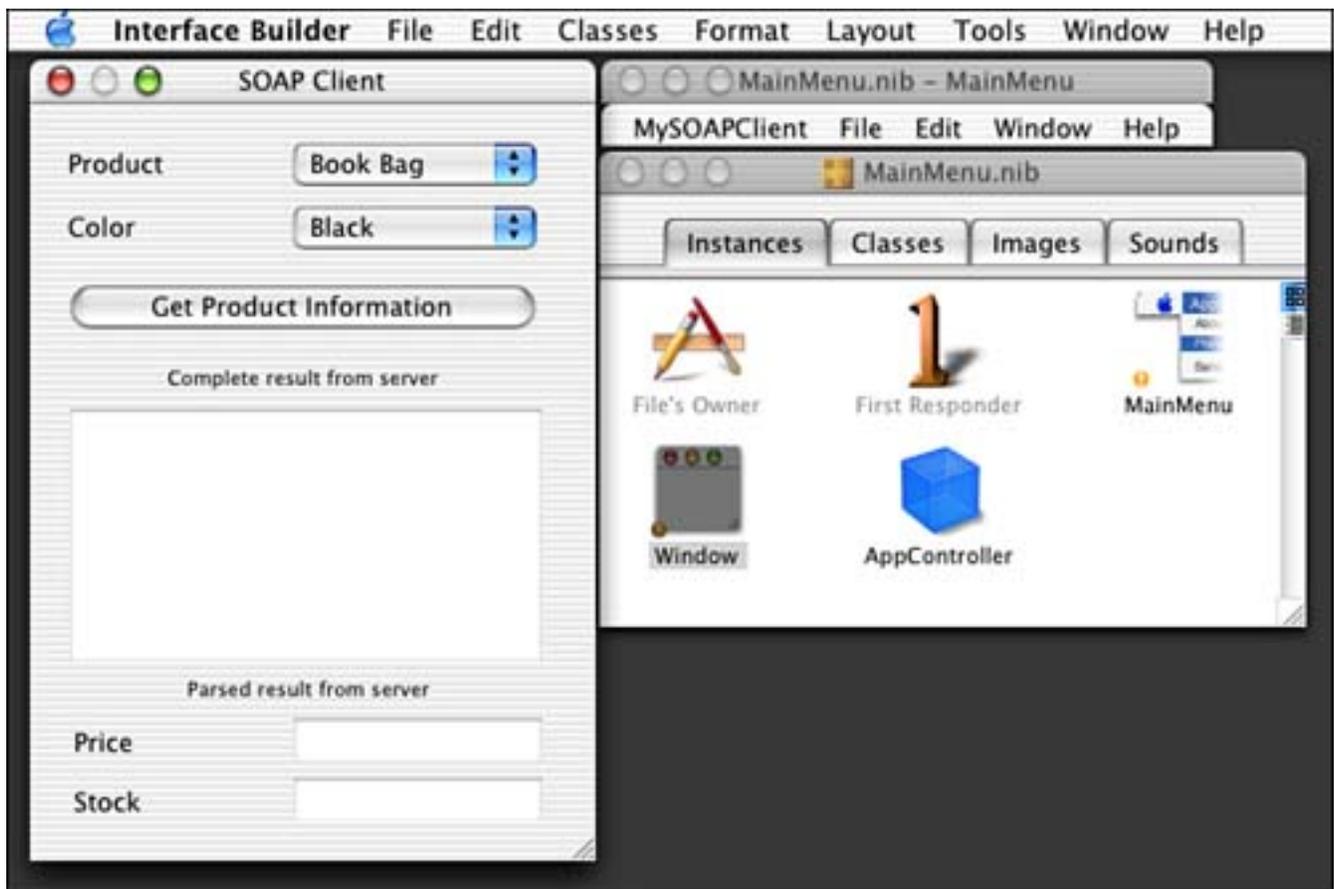
Figure 15.8. The MySOAPClient InfoPlist entries.



The Nib File

The `MainMenu.nib` file is shown in Interface Builder in [Figure 15.9](#). The `AppController` has been instantiated and is the `NSApplication`'s delegate, as you would expect. The Get Product Information button is linked to the `AppController`'s `-getProductInformation:IBAction` method. The `AppController` also contains `IBOutlet`s for the appropriate items in the main window.

Figure 15.9. The MySOAPClient `MainMenu.nib` file.



The Source Code

The interface of `AppController` is shown in [Listing 15.11](#).

Listing 15.11 `AppController` Interface in `AppController.h`

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

@interface AppController : NSObject
{
    IBOutlet NSPopUpButton *m_product;
    IBOutlet NSPopUpButton *m_color;
    IBOutlet NSTextField *m_price;
    IBOutlet NSTextField *m_stock;
    IBOutlet NSTextField *m_result;
}
- (IBAction) getProductInformation: (id) sender;
@end
```

The `AppController` manages the `IBOutlet`s of our user interface for convenience. One `IBAction` is also connected to the Get Product Information button.

[Listing 15.12](#) contains the `-getProductInformation:` method that is called when the Get Product Information button is clicked. The user has already made her choice of product and color. This method alone is the equivalent of our PHP version of the client.

Listing 15.12 - getProductInformation: in ApplicationController.m

```
- (IBAction) getProductInformation: (id) sender
{
    WSMMethodInvocationRef    soapCall;
    NSURL                     *soapURL;
    NSString                  *methodName;
    NSMutableDictionary        *params;
    NSArray                   *paramsOrder;
    NSDictionary              *result;
    NSString                  *selectedProduct;
    NSString                  *selectedColor;

    //
    // 1. Define the location of the SOAP service and method
    //

    // Create the URL to the SOAP service
    soapURL = [NSURL URLWithString:
        @"http://www.triplesoft.com/soap/product_server.php"];

    // Assign the method name to call on the SOAP service
    methodName = @"get_product_info";

    // Create a method invocation
    // First parameter is the URL to the SOAP service
    // Second parameter is the name of the SOAP method to call
    // Third parameter is a constant to specify the SOAP2001 protocol
    soapCall = WSMMethodInvocationCreate((CFURLRef) soapURL,
        (CFStringRef) methodName, kWSSOAP2001Protocol);

    //
    // 2. Set up the parameters to be passed to the SOAP method
    //

    // Get the users choices
    selectedProduct = [m_product titleOfSelectedItem];
    selectedColor = [m_color titleOfSelectedItem];

    // Add the users choices to the dictionary to be passed as parameters
    params = [NSMutableDictionary dictionaryWithCapacity: 2];
    [params setObject:selectedProduct forKey:@"product_name"];
    [params setObject:selectedColor forKey:@"product_color"];

    // Create the array to specify the order of the parameter values
    paramsOrder = [NSArray arrayWithObjects:@"product_name",
        @"product_color", nil];

    // Set the method invocation parameters
    // First parameter is the method invocation created above
    // Second parameter is a dictionary containing the parameters themselves
    // Third parameter is an array specifying the order of the parameters,
    // sometimes optional for SOAP
    WSMMethodInvocationSetParameters(soapCall, (CFDictionaryRef) params,
        (CFArrayRef) paramsOrder);
}
```

```

//
// 3. Make the call and parse the results!
//

// Invoke the method which returns a dictionary of results
result = (NSDictionary*)WSMethodInvocationInvoke(soapCall);

// If the results are a fault, display an error to the user with the fault
// code and descriptive string
if (WSMethodResultIsFault((CFDictionaryRef)result)) {
    NSRunAlertPanel([NSString stringWithFormat:@"Error %@",
        [result objectForKey:(NSString*)kWSFaultCode]],
        [result objectForKey:(NSString*)kWSFaultString], @"OK", @"", @"");
} else {
    // Otherwise, pull the results from the dictionary,
    // held as another dictionary named "soapVal"
    NSDictionary *dictionary = [result objectForKey:
        (NSString*)kWSMethodInvocationResult];
    NSDictionary *soapVal = [dictionary objectForKey:@"soapVal"];

    // Display the entire dictionary result from the server
    [m_result setStringValue: [dictionary description]];

    // Display the specific fields we are interested in (price, stock)
    [m_price setStringValue: [soapVal objectForKey:@"product_price"]];
    [m_stock setStringValue: [soapVal objectForKey:@"product_stock"]];
}

// Release those items that need to be released
[params release];
params = nil;
[paramsOrder release];
paramsOrder = nil;
[result release];
result = nil;
}

```

Let's look at the method in stages.

The first thing we do in [Listing 15.13](#) is to define all the variables we will be using in the method. A lot are listed here, but most are straightforward: `NSArray`s, `NSString`s, and `NSDictionary`s (mutable and otherwise). However, one item is new. The `WSMethodInvocationRef` variable is used to track and manage the object that handles our SOAP call itself. You'll see how this works shortly.

Listing 15.13 - `getProductInformation`: —Part 1 in `AppController.m`

```

- (IBAction) getProductInformation: (id) sender
{
    WSMethodInvocationRef    soapCall;
    NSURL                    *soapURL;
    NSString                  *methodName;
    NSMutableDictionary       *params;
    NSArray                   *paramsOrder;
}

```

```

NSMutableDictionary          *result;
NSString                    *selectedProduct;
NSString                    *selectedColor;

//
// 1. Define the location of the SOAP service and method
//

// Create the URL to the SOAP service
soapURL = [NSURL URLWithString:
    @"http://www.triplesoft.com/soap/product_server.php"];

// Assign the method name to call on the SOAP service
methodName = @"get_product_info";

// Create a method invocation
// First parameter is the URL to the SOAP service
// Second parameter is the name of the SOAP method to call
// Third parameter is a constant to specify the SOAP2001 protocol
soapCall = WSMethodInvocationCreate((CFURLRef) soapURL,
    (CFStringRef) methodName, kWSSOAP2001Protocol);

```

Next, we want to create an `NSURL` that identifies the server that we will be calling. We also want to assign the `methodName` variable to contain the name of the published service we are interested in, `get_product_info`. Last, we create the `WSMethodInvocation` by calling `WSMethodInvocationCreate`, part of the Web Services functionality. This method takes three parameters. The first is the URL of the server. The second is the name of the published service. The third is a constant to specify which protocol to use—in our case, `kWSSOAP2001Protocol`. Other protocol options include XML-RPC. These are all defined in `WSMethodInvocation.h`.

In [Listing 15.14](#), we set up the parameters to be passed to the published service based on the user's choices. First, we obtain the values of the pop-up buttons for product and color as strings. We then create an `NSMutableDictionary` capable of storing two items using the `+dictionaryWithCapacity:` class method. Then, using the `-setObject:forKey:` method of `NSMutableDictionary`, we add the strings to the dictionary.

Listing 15.14 - `getProductInformation:` —Part 2 in `AppController.m`

```

//
// 2. Set up the parameters to be passed to the SOAP method
//

// Get the users choices
selectedProduct = [m_product titleOfSelectedItem];
selectedColor = [m_color titleOfSelectedItem];

// Add the users choices to the dictionary to be passed as parameters
params = [NSMutableDictionary dictionaryWithCapacity: 2];
[params setObject:selectedProduct forKey:@"product_name"];
[params setObject:selectedColor forKey:@"product_color"];

// Create the array to specify the order of the parameter values
paramsOrder = [NSArray arrayWithObjects:@"product_name",
    @"product_color", nil];

```

```

// Set the method invocation parameters
// First parameter is the method invocation created above
// Second parameter is a dictionary containing the parameters themselves
// Third parameter is an array specifying the order of the parameters,
// sometimes optional for SOAP
WSMethodInvocationSetParameters(soapCall, (NSDictionaryRef) params,
    (CFArrayRef) paramsOrder);

```

Although it has been said that SOAP parameters are based on names and this parameter order isn't required, I've found that this isn't necessarily true; therefore, we must create an `NSArray` to specify the parameters and their expected order. We simply add the items in the proper order using the `NSArray+arrayWithObject:` class method. Failure to include this parameter will cause your server to become confused. I am unsure at the time of this writing whether this is a NuSOAP issue or something else—for now, include this information.

We use the `WSMethodInvocationSetParameters` function to attach the parameters to the `WSMethodInvocationRef`. This function takes three parameters. The first is the reference to the method invocation created earlier. The second is the dictionary containing the parameters themselves. The third is the array specifying the order of the parameters.

In [Listing 15.15](#), we make the call to the published service by calling the `WSMethodInvocationInvoke` function and passing the `WSMethodInvocationRef`. This is equivalent to the `$client->call("get_product_info", $params)` method in the PHP client version. This single function handles locating the remote server, transmitting the message to the client via the HTTP protocol, and accepting any resulting data from the server as an `NSDictionary`.

Listing 15.15 - `getProductInformation`:—Part 3 in `AppController.m`

```

//
// 3. Make the call and parse the results!
//

// Invoke the method which returns a dictionary of results
result = (NSDictionary*) WSMethodInvocationInvoke(soapCall);

// If the results are a fault, display an error
// to the user with the fault code
// and descriptive string
if (WSMethodResultIsFault((NSDictionaryRef) result)) {
    NSRunAlertPanel([NSString stringWithFormat:@"Error %@",
        [result objectForKey: (NSString*) kWSFaultCode]],
        [result objectForKey: (NSString*) kWSFaultString], @"OK", @"", @"");
} else {
    // Otherwise, pull the results from the dictionary,
    // held as another dictionary named "soapVal"
    NSDictionary *dictionary = [result objectForKey:
        (NSString*) kWSMethodInvocationResult];
    NSDictionary *soapVal = [dictionary objectForKey:@"soapVal"];

    // Display the entire dictionary result from the server
    [m_result setStringValue: [dictionary description]];

```

```

    // Display the specific fields we are interested in (price, stock)
    [m_price setStringValue: [soapVal objectForKey:@"product_price"]];
    [m_stock setStringValue: [soapVal objectForKey:@"product_stock"]];
}

// Release those items that need to be released
[params release];
params = nil;
[paramsOrder release];
paramsOrder = nil;
[result release];
result = nil;
}

```

We then check the results for an error by calling the `WSMethodResultIsFault` function, passing the `NSDictionary` typecast as a `CFDictionaryRef`. If the function returns `TRUE`, we pull the `kWSFaultCode` and `kWSFaultString` from the result and display them to the user. The `kWSFaultString` is the error returned from the server in the `$err` variable.

Assuming that there are no errors, we pull a dictionary of results from the `WSMethodResultIsFault` return value using the `-objectForKey:` method and passing `kWSMethodInvocationResult`. We must then pull a sub-dictionary of the actual parameters, named `soapVal`, from this dictionary, also using `-objectForKey:`.

After we have completely dereferenced the results, the first thing we do is display the entire dictionary (for debugging purposes) by calling the `-description` method of `NSMutableDictionary`. This allows us to quickly see all fields in the result and display them to the user. Next, we pull just the price and stock information from the sub-dictionary, by name, to display to the user as well.

At the end of the line, we merely clean up after ourselves and release all the memory that needs to be released.

So, what do you think? Using NuSOAP, SOAP isn't entirely complex after all!

[\[Team LiB \]](#)

◀ PREVIOUS NEXT ▶

Try This

A client/server architecture is a great place to experiment with different implementations. Here are some ideas to keep you busy with this one.

Collect the list of product names and colors used in the HTML form in the PHP client from the server itself. Assume that the client knows nothing about what products are available. The server should implement functions named `getProductNames` and `getProductColors` to be listed to the user. Once you get it working in the PHP client, implement it in the Cocoa client as well.

Use MySQL as your data source in the server. If you don't want to explore MySQL, check out the Flat File DataBase (FFDB) PHP Library by John Papandriopoulos. John's library offers a simple way to use database functionality quickly and easily on your server using PHP:

<http://ffdb-php.sourceforge.net/>

Conclusion

SOAP includes many more advanced options than the simple example shown here. If you plan to do any serious SOAP development, be sure to review the SOAP specification online—the URL is listed earlier in this chapter.

Chapter 16. Snippets

"Just one line of code; the hard part is figuring out which one."

—Mike Trent

This chapter is designed to offer a plethora of code snippets and tips in various forms to help you get the most out of your Mac OS X development efforts. You will find various languages represented here. Many of these snippets are demonstrated in the Project Builder project that goes with this chapter. Enjoy!

Calling Carbon from Cocoa

It is very easy to call traditional Carbon functionality from within your Cocoa application. Simply include the framework `Carbon.framework` in your project, `#import <Carbon/Carbon.h>` in the source file you will be calling from, and make the call. Cocoa and Carbon live peacefully together with nary a bad thought for one another.

Drawing with QuickDraw in a Cocoa Application

First, use `NSQuickDrawView`. After the view's focus is locked, you can start calling QuickDraw functions.

Second, you won't get much in the way of clipping for free, as you would when working with Cocoa-native drawing commands, so be sure to perform your own clipping.

Third, when you're finished drawing, you need to flush the window contents out to screen. If you have the `qdPort` handy, you can simply do this:

```
QDFlushPortBuffer(qdPort, nil);
```

Displaying the Username

The `NSFullUserName()` function returns the full username of the currently logged in user. You can also use the `NSUserName()` function to get the short version of the name. See `NSPathUtilities.h` for more useful routines.

Finding a User's Home Directory

The `NSHomeDirectory()` function returns the home directory of the currently logged in user. You can also find the home directory of any username by using the `NSHomeDirectoryForUser(NSString *username)` function. See `NSPathUtilities.h` for more useful routines.

Finding Application Support Directories

In order to find application support directories in the user's **Li brary** directory, the **/Li brary** directory, and the **/Network/Li brary** directory without hard-coding path locations, you should use `NSSearchPathForDirector iesInDomai ns` with the following values. This function returns an `NSArray` of results.

```
typedef enum {

    // supported applications (Applications)
    NSApplicationDirectory = 1,

    // unsupported applications, demonstration versions (Demos)
    NSDemoApplicationDirectory,

    // developer applications (Developer/Applications)
    NSDeveloperApplicationDirectory,

    // system and network administration applications (Administration)
    NSAdminApplicationDirectory,

    // various user-visible documentation, support, and
    // configuration files, resources (Library)
    NSLibraryDirectory,

    // developer resources (Developer)
    NSDeveloperDirectory,

    // user home directories (Users)
    NSUserDirectory,

    // documentation (Documentation)
    NSDocumentationDirectory,

#if MAC_OS_X_VERSION_10_2 <= MAC_OS_X_VERSION_MAX_ALLOWED
    // documents (Documents)
    NSDocumentDirectory,
#endif

    // all directories where applications can occur
    NSAllApplicationsDirectory = 100,

    // all directories where resources can occur
    NSAllLibrariesDirectory = 101

} NSSearchPathDirectory;

typedef enum {

    // user's home directory ---
    // place to install user's personal items (~)
    NSUserDomainMask = 1,
```

```
// local to the current machine --- place to install items
// available to everyone on this machine (/Local)
NSLocalDomainMask = 2,

// publically available location in the local area network
// --- place to install items available on the network (/Network)
NSNetworkDomainMask = 4,

// provided by Apple, unmodifiable (/System)
NSSystemDomainMask = 8,

// all domains: all of the above and future items
NSAllDomainsMask = 0xffff
```

```
} NSSearchPathDomainMask;
```

```
FOUNDATION_EXPORT NSArray *NSSearchPathForDirectoriesInDomains(
    NSSearchPathDirectory directory,
    NSSearchPathDomainMask domainMask,
    BOOL expandTilde);
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Display Alert Panel and Write to Log

The following code shows how to display an alert panel and write the message "The OK button was pressed." if the user presses the OK button. Because these are recently deprecated, you should only use these during debugging and choose sheets (`NSBeginAlertSheet`) instead for anything your users might see.

```
if (NSRunAlertPanel(@"Warning", @"Do you really want to blah?",
    @"OK", @"Cancel", NULL) == NSOKButton)
{
    NSLog(@"The OK button was pressed.");
}
```

In fact, this brings up a very good point. In the time it took to write this example and this book being published, the `NSRunAlertPanel` function became deprecated. Be aware that especially now, while OS X is young and growing, things will change rapidly. You might find an API stripped out from under you. Apple, however, usually gives fair warning, so you have time to find an alternative if one isn't provided for you. Keep this in mind when updating your development environment because you might be in for a surprise now and then as things evolve.

Updating the Application's Dock Image

The following code shows how to update the image that is displayed in the Dock for your application.

```
// Load the default image here  
myImage = [NSImage imageNamed:@"NSApplicationIcon"];  
// Edit the image here (add a badge, etc.)  
// ...  
// Set the edited image in the Dock  
[NSApp setApplicationIconImage: myImage];
```

This can be a useful way to show progress or status of your application. Normally you will not want to alter the image so that it is no longer recognized as your application; instead use this technique sparingly (see [Figure 16.1](#) for an example).

Figure 16.1. An application after changing the dock image programmatically.



See the sample code for this chapter.

Updating the Application's Dock Menu

The following code shows how to update the menu that is displayed in the Dock for your application and respond to the items in it. In your application's delegate, implement the `-applicationDockMenu:` delegate method to return an `NSMenu` that contains your custom items. In this implementation, `m_menu` is an `NSMenu*` instance variable, and we are adding two items to the menu: Beep Once and Alert Once. Don't forget to release `m_menu` when the application quits.

```
- (NSMenu *) applicationDockMenu: (NSApplication *) sender
{
    if (m_menu == nil) {
        m_menu = [[[NSMenu alloc] initWithTitle:
                    NSLocalizedString(@"Beep Once", @"")
                    action:@selector(beepOnce:) keyEquivalent:@""]];
        [m_menu addItem: [[[NSMenuItem alloc] initWithTitle:
                            NSLocalizedString(@"Alert Once", @"")
                            action:@selector(alertOnce:) keyEquivalent:@""]];
    }
    return m_menu;
}
```

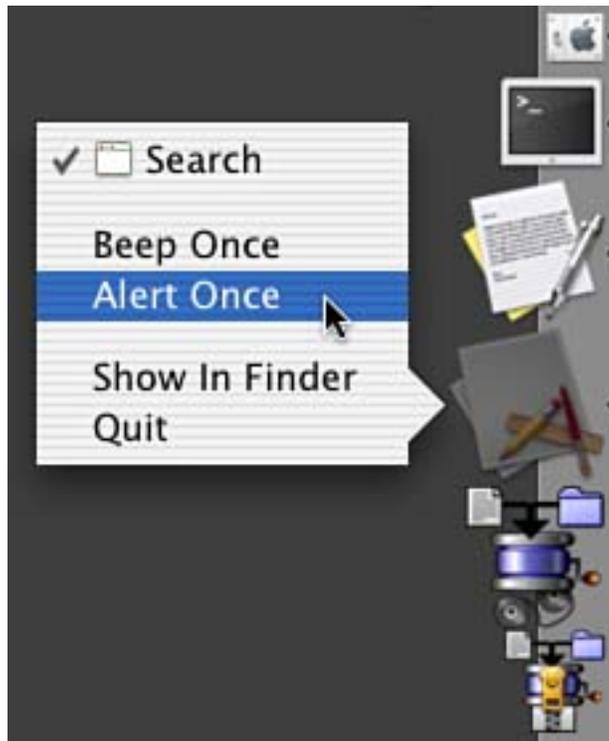
You then implement the action methods as appropriate.

```
- (void) beepOnce: (id) sender
{
    NSBeep();
}

- (void) alertOnce: (id) sender
{
    NSRunAlertPanel(@"Warning", @"You look like blah!",
                    @"OK", NULL, NULL);
}
```

When you control-click (or click and hold) on your application's icon in the Dock, you will see your items appear in the menu. Selecting them calls the appropriate action (see [Figure 16.2](#)).

Figure 16.2. A custom Dock menu.



Because `-applicationDockMenu:` is called each time you control-click on your application's icon in the Dock, you can alter it each time if you like. This makes for a menu that can be dynamically managed depending on the state of your application.

See the sample code for this chapter.

[\[Team LiB \]](#)

Open a URL

The following code shows how to open a URL from within your application. This can be used if you have a Web page that you would like to open in response to a menu item being selected.

```
BOOL success = [[NSWorkspace sharedWorkspace]
    openURL:@"http://www.tripl esoft.com/"];
```

Get the Icon of a File

The following code will retrieve the 32x32 pixel icon of the specified file:

```
NSImage* theImage = [[NSWorkspace sharedWorkspace]
    iconForFile:@" /path/to/file"];
```

Scrolling About Box

To make your application's About box contain a scrolling list of credits, simply add an RTF file named `Credits.rtf` to your project (see [Figure 16.3](#)).

Figure 16.3. A scrolling About box.



Checking Network Status

To check the reachability of a host on a network, you can use the `SCNetworkCheckReachabilityByName` function. Simply replace `www.apple.com` with whatever host you are interested in. You can also use the `SCNetworkCheckReachabilityByAddress` function if you only have an IP address. Simply add `SystemConfiguration.framework` to your project and `#import` the `SystemConfiguration.h` header file.

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

- (BOOL)isNetworkReachable
{
    BOOL          result;
    SCNetworkConnecti onFl ags    fl ags;
    result = SCNetworkCheckReachabilityByName("www.apple.com", &fl ags);
    return (result && (fl ags & kSCNetworkFl agsReachable));
}
```

The `SCNetworkConnecti onFl ags` allow you to verify a number of settings including the following:

`kSCNetworkFl agsTransi entConnecti on`— Indicates that the machine in question can be reached via a transient (that is, PPP) connection.

`kSCNetworkFl agsReachable`— Indicates that the machine in question can be reached using the current network configuration.

`kSCNetworkFl agsConnecti onRequi red`— Indicates that the machine in question can be reached using the current network configuration, but a connection must be established first. For example, a dial-up connection that was not currently active might return this flag.

`kSCNetworkFl agsConnecti onAutomati c`— Indicates that the machine in question can be reached using the current network configuration, but a connection must be established first. Any attempt to pass data to the machine in question will initiate the connection.

`kSCNetworkFl agsInterventi onRequi red`— Indicates that the machine in question can be reached using the current network configuration, but a connection must be established first. In addition, some form of user intervention will be required such as typing in a password, and so on.

Checking for Modifier Keys

You can use the following `category`, found posted on the Net with no author specified, to check for modifier keys at any time during your programs execution. This `category` uses Carbon function calls to check the status of the modifier keys. Simply call `+isOptionKeyDown` to check the status of the option key. The method will return `YES` if the key is down and `NO` otherwise.

```
@interface NSEvent (ModifierKeys)
+ (BOOL) isControlKeyDown;
+ (BOOL) isOptionKeyDown;
+ (BOOL) isCommandKeyDown;
+ (BOOL) isShiftKeyDown;
@end

@implementation NSEvent (ModifierKeys)

+ (BOOL) isControlKeyDown
{
    return (GetCurrentKeyModifiers() & controlKey) != 0;
}

+ (BOOL) isOptionKeyDown
{
    return (GetCurrentKeyModifiers() & optionKey) != 0;
}

+ (BOOL) isCommandKeyDown
{
    return (GetCurrentKeyModifiers() & cmdKey) != 0;
}

+ (BOOL) isShiftKeyDown
{
    return (GetCurrentKeyModifiers() & shiftKey) != 0;
}

@end
```

Executing AppleScripts via URL

Using NSAppleScript's `-initWithContentsOfURL:error:` and `-executeAndReturnError:` methods, you can easily load and execute a text or compiled AppleScript via the Internet from within your application. Think of the possibilities! By storing key portions of the functionality of your application on a Web server, whenever your user executes that functionality, it will be completely up-to-date with any bug fixes you deem necessary. Mind you, accessing code via the Net can become a speed liability, but depending on the need, this could be a very cool application feature!

Defaults

The program named `defaults` is an excellent command-line utility to manipulate user default settings in any application. The syntax is as follows:

```
defaults [-currentHost | -host <hostname>] <subcommand>
```

For example, to change "`MyCompanyName`" that appears at the top of every source file you create with Project Builder, you can use the following command (all on one line) in the terminal:

```
defaults write com.apple.projectbuilder  
'PBXCustomTemplateMacroDefinitions'  
"{ORGANIZATIONNAME = 'Your Company Name Here';}"
```

This command writes an updated value to the `ORGANIZATIONNAME` key of the `PBXCustomTemplateMacroDefinitions` array. Type `man defaults` in the Terminal for complete information.

Find

The `find` command is a very powerful command line for those just getting used to typing their commands. `find` recursively descends the directory tree for each chosen pathname and evaluates an expression for each item in the tree. This means you can quickly and easily search for files that match a certain pattern and perform a task on them. Let's look at some examples:

```
sudo find / -name ".DS_Store" -delete
```

This command deletes all Finder `.DS_Store` files on all disks.

```
find / -name "*.c" -print
```

This command prints all filenames that end in `".c"`.

```
find . -newerct '1 minute ago' -print
```

This command prints all filenames that have changed at the last minute.

```
find /System/Library/Frameworks/Foundation.framework  
-type f -name "*.h" -exec  
grep NSSearchPathForDirectoriesInDomains '{} ' \; -print
```

This command looks at all the headers in Foundation, prints every occurrence of `NSSearchPathForDirectoriesInDomains`, and prints the name of the files in which the function name appears—it should all be typed on one line.

Type `man find` in the Terminal for complete information on this useful tool.

Sample

The `sample` command-line tool allows you to profile a process for a specified time interval. Using `sample`, you can gather data about the running behavior of a process. This utility stops the process at user-defined intervals, samples the current function and the stack, and then enables the application to continue. When all is said and done, you can view a report showing what functions were executing during the sampling. This program is very similar to its GUI cousin, `Sampler.app`.

Type `man sample` in the Terminal for complete information on this useful debugging tool.

Localizing Your Cocoa Application

Every Cocoa application should be localizable. It's not just a good idea; it's easy!

First, you want to use the `NSLocalizedString` macro, or one of its related macros, in your application whenever you reference a string. You simply pass your native language into this macro as follows:

```
NSLocalizedString(@"Cancel ", @"Put a comment here.");
```

Then, you run the `genstrings` command-line tool to generate the `.strings` file for localization. The file would contain entries similar to the following:

```
/* Put a comment here. */  
"Cancel " = "Cancel ";
```

Your translator would then make a copy of this file and change the right-hand side of the `=` sign to be in the translated language. This entry would ultimately look like the following, for Spanish:

```
/* Put a comment here. */  
"Cancel " = "Cancel aci ón";
```

You then simply place the localized files back in your project as localizable resources. That's it! Now when your application makes the call to `NSLocalizedString` as previously, the proper text for the current localization will be returned; English, Spanish, French, and so on.

For more detailed information on localizing, you should visit the following URL:

http://developer.apple.com/techpubs/macosx/Essentials/SystemOverview/International/Localizing_Strings.html

CURLHandle

CURLHandle is a public domain Cocoa class by Dan Wood that subclasses **NSURLHandle** and wraps the **curl** command-line application. This allows you to take advantage of the power of **curl** from within your Cocoa application. Using **curl**, you can easily access network-based data using a variety of protocols including HTTP, HTTPS, GET, POST, FTP, TELNET, LDAP, and more. To learn more about the power of curl, type **man curl** in the Terminal application. You can download **CURLHandle** here: <http://curlhandle.sourceforge.net/>

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Conclusion

I hope you've enjoyed my collection of snippets.

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 17. Conclusion

Thank you! Goodnight!

Well, maybe I can't get off that easy. Thank you for purchasing and reading this book. I hope you have found it interesting and helpful, and that it finds a permanent home on your bookshelf. May it trigger your interests and pull you in a direction you might not have explored otherwise. I also hope that you respect it enough to recommend it to others who might learn from it. I can honestly say that writing it was something I enjoyed very much. I learned a lot in the process about software development under Mac OS X, as well as about myself as a writer.

Whenever I sign up for a project such as this there are periods of excitement, panic, and just about every emotion in between. When you tell your publisher "500 pages? Sure! No problem!" you forget how hard it was to write 500 pages the first time. You soon remember. However, even during those "Oh My Goodness" times, I wouldn't give up because, for me, in writing a book, the reward is the destination and not the journey. Seeing the book on the shelf and in people's hands—that is why I choose to write in the first place. It's an ego boost, an accomplishment, something that took many months of hard work to put together (even though you can do most of it in your pajamas) and finally, there it is! Any writer not on the *New York Times'* bestseller list will tell you that it's not about the money.

Having said that; thank you! Goodnight!

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Part V: Appendixes

[A Source Code](#)

[B Online Resources](#)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Appendix A. Source Code

[Chapter 3—RadarWatcher](#)

[Chapter 4—MyNSBP_App](#)

[Chapter 4—MyNSBP_Desaturate](#)

[Chapter 4—MyNSBP_RemoveColor](#)

[Chapter 4—plistPlugin](#)

[Chapter 5—CFPlugin](#)

[Chapter 5—CFPluginCarbonApp](#)

[Chapter 5—CFPluginCocoaApp](#)

[Chapter 5—Shared](#)

[Chapter 6—MyCarbonFramework](#)

[Chapter 6—MyCocoaFramework](#)

[Chapter 6—MyCocoaApp](#)

[Chapter 7—MyTextService](#)

[Chapter 8—MyPreferencePane](#)

[Chapter 9—MyStatusItem](#)

[Chapter 10—MyScreenEffect](#)

[Chapter 11—MyColorPicker](#)

[Chapter 12—MyThread](#)

[Chapter 13—MyTerminal](#)

[Chapter 14—MyXMLRPC](#)

[Chapter 15—MySOAP](#)

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 3—RadarWatcher

AppController.h

```
#import <Cocoa/Cocoa.h>
@class PreferenceController;

@interface AppController : NSObject
{
    PreferenceController *preferenceController;
}
- (IBAction) showPreferencePanel : (id) sender;

@end
```

AppController.m

```
#import "AppController.h"
#import "PreferenceController.h"

@implementation AppController

+ (void) initialize
{
    // Create the user defaults dictionary
    NSUserDefaults *defaults;
    defaults = [NSUserDefaults standardUserDefaults];

    if ([defaults integerForKey: pkeyInitialized] == 0) {
        // Save our defaults if not already initialized
        [defaults setObject: [NSNumber numberWithInt: 1] forKey: pkeyInitialized];
        [defaults setObject: [NSNumber numberWithInt: NSOnState]
            forKey: pkeyOpenNewWindow];
        [defaults setObject: [NSNumber numberWithInt: NSOnState]
            forKey: pkeyPlaySound];
        [defaults setObject: [NSNumber numberWithInt: NSOnState]
            forKey: pkeyShowAlert];
    }
}

- (BOOL) applicationShouldOpenUntitledFile: (NSApplication *) sender
{
    return ([[NSUserDefaults standardUserDefaults]
        integerForKey: pkeyOpenNewWindow] == NSOnState);
}

- (IBAction) showPreferencePanel: (id) sender
{
    // Create the PreferenceController if it doesn't already exist
    if (!preferenceController)
        preferenceController = [[PreferenceController alloc] init];
}
```

```
    // Display it
    [preferenceController showWindow:self];
}
```

```
- (void) dealloc
{
    // We are done with the PreferenceController, release it
    [preferenceController release];
    [super dealloc];
}
```

@end

main.m

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

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

MyDocument.h

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

```
#import <RadarView.h>
```

```
#define kSoundName                @"Ping"
#define kDefaultURL                @"http://type your weather map url here"
#define kDefaultReloadSeconds      180
#define kMinimumReloadSeconds      10
#define kMaximumReloadSeconds      (60 * 60 * 24)
```

```
#define kNumColors                 5
```

```
extern NSString *keyURL;           // The URL to load
extern NSString *keyReloadSeconds; // The reload interval in seconds
extern NSString *keyWatchBoxRect;  // The watch box rect
extern NSString *keyColors[kNumColors]; // The colors to watch for
extern NSString *keyIgnoreSinglePixels; // TRUE to ignore single pixels
// when analyzing image
extern NSString *keyCloseColors;   // TRUE to consider close colors
// the same when analyzing image
```

```
@interface MyDocument : NSDocument
```

```
{
    NSMutableDictionary *m_md; // MutableDictionary used to read and write data
    NSTimer *m_timer;         // Timer used to reload image regularly

    IBOutlet RadarView *m_radarView; // RadarView displays radar and watch box
    IBOutlet NSTextField *m_statusTextField; // "Last updated" status text
    IBOutlet NSTextField *m_urlTextField; // URL to load
    IBOutlet NSTextField *m_reloadSecondsTextField; // How many seconds to
```

```

// reload image
IBOutlet NSButton *m_startButton; // Start button
IBOutlet NSButton *m_stopButton; // Stop button
IBOutlet NSButton *m_reloadButton; // Reload button
IBOutlet NSButton *m_resetButton; // Reset button (resets colors to
100%
// opaque - ie: clearColor)

IBOutlet NSColorWell *m_colorWell1; // Color wells of colors to look for
IBOutlet NSColorWell *m_colorWell2;
IBOutlet NSColorWell *m_colorWell3;
IBOutlet NSColorWell *m_colorWell4;
IBOutlet NSColorWell *m_colorWell5;

IBOutlet NSButton *m_ignoreSinglePixelsButton; // Ignore single pixels,
// only look for married ones
IBOutlet NSButton *m_closeColorsButton; // Colors that are close to one
// another are considered the same
}

- (void)updateUI; // Called to update the UI components
- (void)destroyTimer; // Called to destroy the current NSTimer
// before a new one is created

// Display an alert over the document window
- (void)displayAlert:(NSString*)title msg:
(NSString*)msg defaultButton:(NSString*)defaultButton;

- (IBAction)refreshRadar:(id)sender; // Called from Reload button
- (IBAction)startRadar:(id)sender; // Called from Start button
- (IBAction)stopRadar:(id)sender; // Called from Stop button
- (IBAction)resetColors:(id)sender; // Called from Reset button
- (IBAction)changeColor:(id)sender; // Called when a color is
// changed in the color picker
- (IBAction)ignoreSinglePixels:(id)sender; // Called when the Ignore
// Single Pixels checkbox is pressed
- (IBAction)closeColors:(id)sender; // Called when the
// Close Colors checkbox is pressed

@end

```

MyDocument.m

```

#import "MyDocument.h"
#import "PreferenceController.h"

// MutableDictionary keys
NSString *keyURL = @"keyURL";
NSString *keyReloadSeconds = @"keyReloadSeconds";
NSString *keyWatchBoxRect = @"keyWatchBoxRect";
NSString *keyColors[kNumColors] = {"keyColors1", "keyColors2",
@"keyColors3", "keyColors4", "keyColors5"};
NSString *keyIgnoreSinglePixels = @"keyIgnoreSinglePixels";
NSString *keyCloseColors = @"keyCloseColors";

@implementation MyDocument

```

```

- (id)init
{
    [super init];
    if (self) {

        // Add your subclass-specific initialization here.
        // If an error occurs here, send a [self dealloc]
        // message and return nil.

        // Create the dictionary that contains our documents
        // data with default values
        m_md = [[NSMutableDictionary dictionary] retain];
        if (m_md) {
            [m_md setObject:kDefaultURL forKey:keyURL];
            [m_md setObject:[NSNumber numberWithInt:kDefaultReloadSeconds]
                forKey:keyReloadSeconds];
            [m_md setObject:[NSStringFromRect(NSMakeRect(0, 0, 0, 0))
                forKey:keyWatchBoxRect];
            [m_md setObject:[NSArchiver archivedDataWithRootObject:
                [NSColor clearColor]]
                forKey:keyColors[0]];
            [m_md setObject:[NSArchiver archivedDataWithRootObject:
                [NSColor clearColor]]
                forKey:keyColors[1]];
            [m_md setObject:[NSArchiver archivedDataWithRootObject:
                [NSColor clearColor]]
                forKey:keyColors[2]];
            [m_md setObject:[NSArchiver archivedDataWithRootObject:
                [NSColor clearColor]]
                forKey:keyColors[3]];
            [m_md setObject:[NSArchiver archivedDataWithRootObject:
                [NSColor clearColor]]
                forKey:keyColors[4]];
            [m_md setObject:[NSNumber numberWithInt:NSOffState]
                forKey:keyIgnoreSinglePixels];
            [m_md setObject:[NSNumber numberWithInt:NSOffState]
                forKey:keyCloseColors];
        } else {
            NSRunAlertPanel(@"Error!",
                @"There was an error allocating our NSMutableDictionary.",
                @"OK", nil, nil);
            [self dealloc];
            return nil;
        }
    }
    return self;
}

- (void)dealloc {
    // Release the memory used by the dictionary
    [m_md release];
    m_md = nil;

    // Stop the timer
    [self destroyTimer];
}

```

```

    // Psst? BlahBl ahBl ah! Pass it on.
    [super dealloc];
}

- (NSString *)windowNibName
{
    // Override returning the nib file name of the document
    // If you need to use a subclass of NSWindowController
    // or if your document supports multiple NSWindowControllers,
    // you should remove this method and override -
    // makeWindowControllers instead.
    return @"MyDocument";
}

- (void)windowControllerDidLoadNib: (NSWindowController *) aController
{
    [super windowControllerDidLoadNib: aController];
    // Add any code here that need to be executed once the windowController
    // has loaded the document's window.

    // First time through we need to tell the RadarView who is the boss
    [m_radarView setDocument: self];

    // Update the UI fields
    [self updateUI];

    // First time through we want to blank out the status text
    [m_statusTextField setStringValue:@""];
}

// Validate our Document menu items
- (BOOL)validateMenuItem: (NSMenuItem *) menuItem
{
    NSString *selectorString;
    selectorString = NSStringFromSelector([menuItem action]);

    NSLog(@"validateMenuItem called for %@", selectorString);
    if ([menuItem action] == @selector(refreshRadar:))
        return YES;
    if ([menuItem action] == @selector(startRadar:))
        return (m_timer == nil);
    if ([menuItem action] == @selector(stopRadar:))
        return (m_timer != nil);

    return [super validateMenuItem: menuItem];
}

- (BOOL)readFromFile: (NSString *) fileName ofType: (NSString *) docType
{
    // Stop the current run if we are loading a new file into this document
    [NSApp sendAction:@selector(stopRadar:) to: self from: self];

    // Release the current dictionary
    [m_md release];
    m_md = nil;
}

```

```

// Load the new data from the file
m_md = [[NSMutableDictionary dictionaryWithContentsOfFile: fileName]
retain];
if (m_md) {
    // Update the UI in case this is a Revert
    [self updateUI];

    // Return a positive result
    return YES;
}

```

```

return NO; // Failure
}

```

```

- (BOOL)writeToFile:(NSString *)fileName ofType:(NSString *)type
{

```

```

    // Update data that needs special conversion

```

```

    // NSRect are best off saved as string representations
    // so the developer can edit them
    [m_md setObject: NSStringFromRect([m_radarView watchBoxRect])
    forKey: keyWatchBoxRect];

```

```

    // NSColor should save but do not, so we must archive
    // and unarchive them to get them to work
    [m_md setObject: [NSArchiver archivedDataWithRootObject:
    [m_colorWell1 color]]
    forKey: keyColors[0]];

```

```

    [m_md setObject: [NSArchiver archivedDataWithRootObject:
    [m_colorWell2 color]]
    forKey: keyColors[1]];

```

```

    [m_md setObject: [NSArchiver archivedDataWithRootObject:
    [m_colorWell3 color]]
    forKey: keyColors[2]];

```

```

    [m_md setObject: [NSArchiver archivedDataWithRootObject:
    [m_colorWell4 color]]
    forKey: keyColors[3]];

```

```

    [m_md setObject: [NSArchiver archivedDataWithRootObject:
    [m_colorWell5 color]]
    forKey: keyColors[4]];

```

```

    // Retrieve the state of the checkbox
    [m_md setObject: [NSNumber numberWithInt:
    [m_ignoreSinglePixelButton state]]
    forKey: keyIgnoreSinglePixels];
    [m_md setObject: [NSNumber numberWithInt: [m_closeColorsButton state]]
    forKey: keyCloseColors];

```

```

    // Write the current dictionary to the file
    return [m_md writeToFile: fileName atomically: YES];
}

```

```

- (void)updateUI
{

```

```

    // Update the UI with the data from the latest file or revert action

```

```

[m_urlTextField setStringValue:[m_md objectForKey:keyURL]];
[m_reloadSecondsTextField setStringValue:
    [m_md objectForKey:keyReloadSeconds]];

// NSColor should save but do not, so we must archive
// and unarchive them to get them to work
[m_colorWell1 setColor:[NSUnarchiver unarchiveObjectWithData:[m_md
    objectForKey:keyColors[0]]]];
[m_colorWell2 setColor:[NSUnarchiver unarchiveObjectWithData:[m_md
    objectForKey:keyColors[1]]]];
[m_colorWell3 setColor:[NSUnarchiver unarchiveObjectWithData:[m_md
    objectForKey:keyColors[2]]]];
[m_colorWell4 setColor:[NSUnarchiver unarchiveObjectWithData:[m_md
    objectForKey:keyColors[3]]]];
[m_colorWell5 setColor:[NSUnarchiver unarchiveObjectWithData:[m_md
    objectForKey:keyColors[4]]]];

// Set the state of the checkbox
[m_ignoreSinglePixelsButton setState:[m_md objectForKey:
    keyIgnoreSinglePixels] intValue];
[m_closeColorsButton setState:[m_md objectForKey:
    keyCloseColors] intValue];

// Tell the radar view what the current watch box
// rectangle is so it can draw it properly
[m_radarView setWatchBoxRect:NSRectFromString(
    [m_md objectForKey:keyWatchBoxRect]);
}

```

```

- (void)controlTextDidChange:(NSNotification*) aNotification
{
    // This function is called when the user types a character
    // in the URL or Reload Seconds text fields

    // A change has been made to the document
    [self updateChangeCount:NSChangeDone];

    // Update the proper dictionary entry depending on what changed
    id sender = [aNotification object];
    if (sender == m_urlTextField) {
        [m_md setObject:[m_urlTextField stringValue] forKey:keyURL];
    } else if (sender == m_reloadSecondsTextField) {
        [m_md setObject:[NSNumber numberWithInt:
            [m_reloadSecondsTextField intValue]]
            forKey:keyReloadSeconds];
    }
}

```

```

- (IBAction)changeColor:(id) sender
{
    // This function is called when the user changes a colorwell

    // A change has been made to the document
    [self updateChangeCount:NSChangeDone];
}

```

```

- (IBAction)refreshRadar: (id) sender
{
    // Load the current image from the URL and display it on the screen,
    // releasing any previous image
    // Note we must use this method and not [UIImage initWithContentsOfURL]
    // because it uses the cache
    NSData *data = [[NSURL URLWithString: [m_md objectForKey: keyURL]]
        resourceDataUsingCache: NO];
    UIImage *image = [[UIImage alloc] initWithData: data];

    // If we loaded an image...
    if (image) {

        // Pass it to the RadarView for display
        [m_radarView setImage: image];

        // We no longer need the image,
        // m_radarView has "retained" it at this point
        [image release];

        // Update the current time as the last time we loaded the image
        [m_statusTextField setStringValue:
            [NSString localizedStringWithFormat: @"Map last loaded %@",
            [NSDate date]]];

        // Check for colors
        BOOL ignoreSinglePixelState =
            ([m_ignoreSinglePixelButton state] == NSOnState);
        BOOL closeColorsState = ([m_closeColorsButton state] == NSOnState);
        if (([[m_colorWell1 color] alphaComponent] > 0) &&
            ([m_radarView isColorInImageInRect: [m_colorWell1 color]
            ignoreSinglePixelState: ignoreSinglePixelState
            closeColors: closeColorsState] == YES)) {
            [self displayAlert:@"Found Color 1!"
                msg:@"A color was found in the watch box."
                defaultButton:@"OK"];
        }
        if (([[m_colorWell2 color] alphaComponent] > 0) &&
            ([m_radarView isColorInImageInRect: [m_colorWell2 color]
            ignoreSinglePixelState: ignoreSinglePixelState
            closeColors: closeColorsState] == YES)) {
            [self displayAlert:@"Found Color 2!"
                msg:@"A color was found in the watch box."
                defaultButton:@"OK"];
        }
        if (([[m_colorWell3 color] alphaComponent] > 0) &&
            ([m_radarView isColorInImageInRect: [m_colorWell3 color]
            ignoreSinglePixelState: ignoreSinglePixelState
            closeColors: closeColorsState] == YES)) {
            [self displayAlert:@"Found Color 3!"
                msg:@"A color was found in the watch box."
                defaultButton:@"OK"];
        }
        if (([[m_colorWell4 color] alphaComponent] > 0) &&
            ([m_radarView isColorInImageInRect: [m_colorWell4 color]
            ignoreSinglePixelState: ignoreSinglePixelState

```

```

        closeColors: closeColorsState] == YES)) {
            [self displayAlert:@"Found Color 4!"
                msg:@"A color was found in the watch box."
                defaultButton:@"OK"];
        }
        if ((([m_colorWell5 color] alphaComponent] > 0) &&
            ([m_radarView isColorInImageInRect: [m_colorWell5 color]
             ignoreSinglePixels: ignoreSinglePixelsState
             closeColors: closeColorsState] == YES)) {
            [self displayAlert:@"Found Color 5!"
                msg:@"A color was found in the watch box."
                defaultButton:@"OK"];
        }
    } else {
        NSBeep();
    }
}

```

```
// The user pressed the Start button
```

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

```
{
```

```

    [m_stopButton setEnabled: TRUE];
    [m_startButton setEnabled: FALSE];
    [m_resetButton setEnabled: FALSE];
    [m_colorWell1 setEnabled: FALSE];
    [m_colorWell2 setEnabled: FALSE];
    [m_colorWell3 setEnabled: FALSE];
    [m_colorWell4 setEnabled: FALSE];
    [m_colorWell5 setEnabled: FALSE];
    [m_urlTextField setEnabled: FALSE];
    [m_reloadSecondsTextField setEnabled: FALSE];
    [m_ignoreSinglePixelsButton setEnabled: FALSE];
    [m_closeColorsButton setEnabled: FALSE];
    [m_radarView setLocked: TRUE];

```

```
// Verify refresh time limits
```

```
if ([m_md objectForKey: keyReloadSeconds] intValue] <
```

```
    kMinimumReloadSeconds) {
```

```

        [m_md setObject: [NSNumber numberWithInt: kMinimumReloadSeconds]
            forKey: keyReloadSeconds];

```

```

        [m_reloadSecondsTextField setStringValue:
            [m_md objectForKey: keyReloadSeconds]];

```

```

        NSRunAlertPanel(@"Error!",
            @"Minimum refresh time is 10 seconds; I fixed it.",
            @"OK", nil, nil);
    }

```

```
if ([m_md objectForKey: keyReloadSeconds] intValue] >
```

```
    kMaximumReloadSeconds) {
```

```

        [m_md setObject: [NSNumber numberWithInt: kMaximumReloadSeconds]
            forKey: keyReloadSeconds];

```

```

        [m_reloadSecondsTextField setStringValue:
            [m_md objectForKey: keyReloadSeconds]];

```

```

        NSRunAlertPanel(@"Error!",
            @"Maximum refresh time is 1 days worth of seconds; I fixed it.",

```

```

        @"OK", nil, nil);
    }

    // Stop the timer
    [self destroyTimer];

    // Refresh the image immediately
    [NSApp sendAction:@selector(refreshRadar:) to:self from:self];

    // Set up our timer to execute every keyReloadSeconds seconds
    m_timer = [[NSTimer scheduledTimerWithTimeInterval:
        (NSTimeInterval)[[m_md objectForKey:keyReloadSeconds] doubleValue]
        target:self selector:@selector(refreshRadar:)
        userInfo:nil repeats:YES] retain];
    if (m_timer == nil) {
        NSRunAlertPanel(@"Error!",
            @"There was an error allocating our NSTimer.",
            @"OK", nil, nil);
    }
}

// The user pressed the Stop button
- (IBAction) stopRadar: (id) sender
{
    [m_stopButton setEnabled:FALSE];
    [m_startButton setEnabled:TRUE];
    [m_resetButton setEnabled:TRUE];
    [m_colorWell1 setEnabled:TRUE];
    [m_colorWell2 setEnabled:TRUE];
    [m_colorWell3 setEnabled:TRUE];
    [m_colorWell4 setEnabled:TRUE];
    [m_colorWell5 setEnabled:TRUE];
    [m_urlTextField setEnabled:TRUE];
    [m_reloadSecondsTextField setEnabled:TRUE];
    [m_ignoreSignalPixelButton setEnabled:TRUE];
    [m_closeColorsButton setEnabled:TRUE];
    [m_radarView setLocked:FALSE];

    // Stop the timer
    [self destroyTimer];
}

// The user pressed the Reset button
- (IBAction) resetColors: (id) sender
{
    int i = NSRunAlertPanel(@"Reset Colors?",
        @"Reset all the colors to their default values?", @"No", @"Yes", nil);
    if (i == NSAlertAlternateReturn) {
        // Reset colors
        [m_colorWell1 setColor:[NSColor clearColor]];
        [m_colorWell2 setColor:[NSColor clearColor]];
        [m_colorWell3 setColor:[NSColor clearColor]];
        [m_colorWell4 setColor:[NSColor clearColor]];
        [m_colorWell5 setColor:[NSColor clearColor]];

        [self updateChangeCount:NSChangeDone];
    }
}

```

```

    }
}

// Called when the Ignore Single Pixels checkbox is pressed
- (IBAction)ignoreSinglePixels:(id)sender
{
    [self updateChangeCount: NSChangeDone];
}

// Called when the Close Colors checkbox is pressed
- (IBAction)closeColors:(id)sender
{
    [self updateChangeCount: NSChangeDone];
}

// Shorthand to destroy the current timer
- (void)destroyTimer
{
    if (m_timer != nil) {
        [m_timer invalidate];
        [m_timer release];
        m_timer = nil;
    }
}

// Display a generic OK alert and play a sound
- (void)displayAlert:(NSString*)title msg:
    (NSString*)msg defaultButton:(NSString*)defaultButton
{
    // Play a sound if our preferences say so
    if ([[NSUserDefaults standardUserDefaults]
        integerForKey:pkeyPlaySound] == NSOnState)
        [[NSSound soundNamed:kSoundName] play];

    // Show an alert no matter what
    if ([[NSUserDefaults standardUserDefaults]
        integerForKey:pkeyShowAlert] == NSOnState)
        NSBeginAlertSheet(title, defaultButton, nil, nil,
            [[[self windowControllers] objectAtIndex:0] window],
            self, NULL, NULL, NULL, msg);
}

@end

```

PreferenceController.h

```

#import <AppKit/AppKit.h>

extern NSString *pkeyInitialized;
extern NSString *pkeyOpenNewWindow;
extern NSString *pkeyPlaySound;
extern NSString *pkeyShowAlert;

@interface PreferenceController : NSWindowController {
    IBOutlet NSButton *m_newWindowButton;
}

```

```

IBOutlet NSButton *m_playSoundButton;
IBOutlet NSButton *m_showAlertButton;
}
- (IBAction) changeNewWindowButton: (id) sender;
- (IBAction) changePlaySoundButton: (id) sender;
- (IBAction) changeShowAlertButton: (id) sender;

@end

```

PreferenceController.m

```
#import "PreferenceController.h"
```

```

NSString *pkeyInitialized = @"pkeyInitialized";
NSString *pkeyOpenNewWindow = @"pkeyOpenNewWindow";
NSString *pkeyPlaySound = @"pkeyPlaySound";
NSString *pkeyShowAlert = @"pkeyShowAlert";

```

```
@implementation PreferenceController
```

```

- (id) init
{
    // Load the Preferences.nib file
    self = [super initWithWindowNibName:@"Preferences"];
    return self;
}

- (void) windowDidLoad
{
   NSUserDefaults *defaults;

    // Load our default values and set the preference controls accordingly
    defaults = [NSUserDefaults standardUserDefaults];
    [m_newWindowButton setState: [defaults integerForKey: pkeyOpenNewWindow]];
    [m_playSoundButton setState: [defaults integerForKey: pkeyPlaySound]];
    [m_showAlertButton setState: [defaults integerForKey: pkeyShowAlert]];
}

- (IBAction) changeNewWindowButton: (id) sender
{
    // Save back the new value of this control to the defaults
    [[NSUserDefaults standardUserDefaults] setInteger: [sender state]
    forKey: pkeyOpenNewWindow];
}

- (IBAction) changePlaySoundButton: (id) sender
{
    // Save back the new value of this control to the defaults
    [[NSUserDefaults standardUserDefaults] setInteger: [sender state]
    forKey: pkeyPlaySound];
}

- (IBAction) changeShowAlertButton: (id) sender
{
    // Save back the new value of this control to the defaults

```

```
[[NSUserDefaults standardUserDefaults] setInteger:[sender state]
    forKey:pkeyShowAlert];
```

```
}
```

```
@end
```

```
RadarView.h
```

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

```
@interface RadarView : NSView
```

```
{
```

```
    NSImage      *m_image;           // The radar image
    NSRect        m_watchBoxRect;     // The watch box
    NSPoint       m_downPoint;        // The last mouse down-point
    NSPoint       m_currentPoint;     // The last mouse current-point
    NSDocument    *m_document;        // The document that owns us
    BOOL          m_locked;           // TRUE if we should disallow watch box drawing
```

```
}
```

```
- (BOOL) isColorInImageInRect: (NSColor *) color
    ignoreSinglePixels: (BOOL) ignore
    closeColors: (BOOL) close;
- (void) setDocument: (NSDocument *) document;
- (void) setImage: (NSImage *) newImage;
- (NSImage*) image;
- (void) setWatchBoxRect: (NSRect) rect;
- (NSRect) watchBoxRect;
- (void) setLocked: (BOOL) locked;
- (BOOL) locked;
```

```
@end
```

```
RadarView.m
```

```
#import "RadarView.h"
```

```
@implementation RadarView
```

```
- (id) initWithFrame: (NSRect) frameRect
```

```
{
```

```
    [super initWithFrame: frameRect];
```

```
    m_image = nil;
    m_watchBoxRect = NSMakeRect(0, 0, 0, 0);
    m_downPoint = NSMakePoint(0, 0);
    m_currentPoint = NSMakePoint(0, 0);
    m_document = nil;
    m_locked = FALSE;
```

```
    return self;
```

```
}
```

```
- (void) dealloc
```

```
{
```

```

    [m_image release];
    [super dealloc];
}

- (void)mouseDown: (NSEvent *) event
{
    if (m_locked) {
        NSRunAlertPanel(@"Locked!",

            @"Please press the Stop button before attempting to draw the watch box.",
            @"OK", nil, nil);
    } else if (m_image) {
        NSPoint p = [event locationInWindow];
        m_downPoint = [self convertPoint:p fromView:nil];
        m_currentPoint = m_downPoint;
        [self setNeedsDisplay:YES];
        [m_document updateChangeCount:NSChangeDone];
    } else {
        NSRunAlertPanel(@"No Image!",

            @"Please press the Reload button once to load the image before
            attempting to draw the watch box.",
            @"OK", nil, nil);
    }
}

- (void)mouseDragged: (NSEvent *) event
{
    if (m_image && !m_locked) {
        NSPoint p = [event locationInWindow];
        m_currentPoint = [self convertPoint:p fromView:nil];
        [[self superview] autoscroll:event];
        [self setNeedsDisplay:YES];
    }
}

- (void)mouseUp: (NSEvent *) event
{
    if (m_image && !m_locked) {
        NSPoint p = [event locationInWindow];
        m_currentPoint = [self convertPoint:p fromView:nil];
        [self setNeedsDisplay:YES];
    }
}

- (NSRect)currentRect
{
    // Calculate the current watch box rectangle based on the mouse points
    float minX = MIN(m_downPoint.x, m_currentPoint.x);
    float maxX = MAX(m_downPoint.x, m_currentPoint.x);
    float minY = MIN(m_downPoint.y, m_currentPoint.y);
    float maxY = MAX(m_downPoint.y, m_currentPoint.y);
    return NSMakeRect(minX, minY, maxX-minX, maxY-minY);
}

- (void)drawRect: (NSRect) rect
{

```

```

if (m_image) {
    // Resize view to be the size of the image
    // to avoid unnecessary scrolling
    [self setFrameSize: [m_image size]];

    // Draw image
    NSRect bounds = [self bounds];
    NSPoint p = bounds.origin;
    [m_image dissolveToPoint: p fraction: 1.0];

    // Draw watch box
    m_watchBoxRect = [self currentRect];
    if (!NSIsEmptyRect(m_watchBoxRect)) {
#ifdef DRAW_WITH_TINT
        [[NSColor lightGrayColor] colorWithAlphaComponent: 0.2] set];
        NSRectFillUsingOperation(m_watchBoxRect, NSCompositeSourceOver);
#endif
        [[NSColor grayColor] set];
        NSFrameRectWithWidth(m_watchBoxRect, 2);
        [[NSColor blackColor] set];
        NSFrameRectWithWidth(m_watchBoxRect, 1);
    }

    // Force an update
    [self setNeedsDisplay: YES];
}
}

```

```

// InRange function returns true if a and b are within r of one another
// TODO: implement "wrap" where range is 0-359 and 360=0, for example.

```

```

FOUNDATION_STATIC_INLINE BOOL InRange(float a, float b, float r) {
    return (((b <= a+r) && (b >= a-r)) &&
            ((a <= b+r) && (a >= b-r))) ;
}

```

```

- (BOOL) isColorInImageInRect: (NSColor *) color
  ignoreSinglePixels: (BOOL) ignore
  closeColors: (BOOL) close

```

```

// Walk each pixel in the watch box and see
// if the passed in color exists within it
// ignoreSinglePixels - if TRUE, must be more
// than one pixel of a color in the watch box,
// does not need to be adjacent to each other
// closeColors - if TRUE, colors match if they
// are "close" (hueComponent (0-359) within
// 10 degrees either way, brightnessComponent (0-100) within 20%)
{

```

```

    int    x, y;
    long   found = 0;

```

```

    // Lock the image focus so we can perform direct pixel reads
    [m_image lockFocus];

```

```

    // Update the watch box rect

```

```

m_watchBoxRect = [self currentRect];

// For each pixel within the watch box rect,
// see if the color matches the color
// we passed into this function
for (x = m_watchBoxRect.origin.x;
     x < m_watchBoxRect.origin.x + m_watchBoxRect.size.width; ++x) {
    for (y = m_watchBoxRect.origin.y;
         y < m_watchBoxRect.origin.y + m_watchBoxRect.size.height;
         ++y) {
        NSColor *newColor = NSReadPixel(NSMakePoint(x, y));
        if (close) {
            if ((InRange([color hueComponent],
                          [newColor hueComponent], .036)) &&
                (InRange([color brightnessComponent],
                          [newColor brightnessComponent], .05))) {
                found++;
            }
        } else {
            if (([color redComponent] == [newColor redComponent]) &&
                ([color greenComponent] == [newColor greenComponent]) &&
                ([color blueComponent] == [newColor blueComponent])) {
                found++;
            }
        }
    }
}

```

```

// Unlock the image focus and return the result of our search
[m_image unlockFocus];
return (ignore ? (found>1) : (found>0));
}

```

```

- (void)setDocument:(NSDocument *)document
{
    m_document = document;
}

```

```

- (void)setImage:(NSImage *)newImage
{
    [newImage retain];
    [m_image release];
    m_image = newImage;
    [self setNeedsDisplay:YES];
}

```

```

- (NSImage*)image
{
    return m_image;
}

```

```

- (void)setWatchBoxRect:(NSRect) rect
{
    m_downPoint.x = rect.origin.x;
    m_downPoint.y = rect.origin.y;
}

```

```
m_currentPoint.x = rect.origin.x + rect.size.width;
m_currentPoint.y = rect.origin.y + rect.size.height;
m_watchBoxRect = [self currentRect];
}

- (NSRect) watchBoxRect
{
    return m_watchBoxRect;
}

- (void) setLocked: (BOOL) locked
{
    m_locked = locked;
}

- (BOOL) locked
{
    return m_locked;
}

@end
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 4—MyNSBP_App

MyNSBP_Protocol.h

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

@protocol MyNSBP_Protocol

+ (BOOL) initializePluginClass: (NSBundle *) theBundle;
+ (void) terminatePluginClass;
+ (id) instantiatePlugin;
- (NSImage*) doPlugin: (NSImage*) theImage;

@end
```

AppController.h

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

@interface AppController : NSObject
{
}
@end
```

AppController.m

```
#import "AppController.h"

@implementation AppController

- (id) init
{
    [super init];
    if (self) {

    }
    return self;
}

- (void) dealloc
{
    [super dealloc];
}

// Causes the application to quit when the one and only window is closed
- (BOOL) applicationShouldTerminateAfterLastWindowClosed: (NSApplication *) sender
{
    return TRUE;
}
```

```
@end
```

```
ImageWindowController.h
```

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

```
@interface ImageWindowController : NSObject
```

```
{
```

```
IBOutlet NSMenuItem *m_filterMenuItem;
```

```
IBOutlet NSImageView *m_imageView;
```

```
NSMutableArray * m_pluginClasses;
```

```
NSMutableArray * m_pluginInstances;
```

```
}
```

```
- (IBAction) doFilter: (id) sender;
```

```
- (IBAction) doResetImage: (id) sender;
```

```
@end
```

```
ImageWindowController.m
```

```
#import "ImageWindowController.h"
```

```
#import "MyNSBP_Protocol.h"
```

```
@implementation ImageWindowController
```

```
- (id) init
```

```
{
```

```
self = [super init];
```

```
if (self) {
```

```
    // Allocate our arrays
```

```
    m_pluginClasses = [[NSMutableArray alloc] init];
```

```
    m_pluginInstances = [[NSMutableArray alloc] init];
```

```
}
```

```
return self;
```

```
}
```

```
- (void) dealloc
```

```
{
```

```
    // Release our arrays
```

```
    [m_pluginClasses release];
```

```
    [m_pluginInstances release];
```

```
    [super dealloc];
```

```
}
```

```
// Called to initialize our fields if we need to -
```

```
// this is where we will load our plugins
```

```
- (void) awakeFromNib
```

```
{
```

```
    NSMenu *filterMenu = [m_filterMenuItem submenu];
```

```
    NSString *folderPath;
```

```
    // Locate the plugins directory within our app package
```



```

        [m_pluginClasses addObject: pluginClass];

        // Add the instance to our array
        [m_pluginInstances addObject: thePlugin];
    }
}

// Update our self image
[self doResetImage:self];
}

// Called when the window is about to close
- (void)windowWillClose: (NSNotification*) notification
{
    Class pluginClass;
    NSEnumerator *enumerator;
    NSObject<MyNSBP_Protocol> *thePlugin;

    // Enumerate through the instances and release each
    enumerator = [m_pluginInstances objectEnumerator];
    while ((thePlugin = [enumerator nextObject])) {
        [thePlugin release];
        thePlugin = nil;
    }

    // Enumerate through the classes and terminate each
    enumerator = [m_pluginClasses objectEnumerator];
    while ((pluginClass = [enumerator nextObject])) {

        [pluginClass terminatePluginClass];
        pluginClass = nil;
    }
}

// Called when one of the plugins is selected from the Filter menu
- (IBAction)doFilter: (id) sender
{
    NSObject<MyNSBP_Protocol>* thePlugin;

    // Get the proper plugin instance from the array based on tag value
    thePlugin = [m_pluginInstances objectAtIndex: [sender tag]];
    if (thePlugin) {
        // Pass the image to the plugin, which will alter it and pass it back
        NSImage *theAlteredImage = [thePlugin doPlugin: [m_imageView image]];

        // Set the altered image to be the current image
        [m_imageView setImage: theAlteredImage];

        // Update the view
        [m_imageView setNeedsDisplay: YES];
    }
}

```

```
// Called when the user pressed the Reset Image
// button, restores the image in the view
// We must make a copy otherwise the actual
// picture.tiff tends to get "edited"
- (IBAction)doResetImage:(id)sender
{
    NSImage *theImage = [[[NSImage imageNamed:
        @"picture.tiff"] copy] autorelease];
    [m_imageView setImage:theImage];
//    [m_imageView setNeedsDisplay:YES];
}
```

@end

main.m

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

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

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 4—MyNSBP_Desaturate

MyNSBP_Desaturate.h

```
#import <Foundation/Foundation.h>
#import "MyNSBP_Protocol.h"

@interface MyNSBP_Desaturate : NSObject<MyNSBP_Protocol> {
}
@end
```

MyNSBP_Desaturate.m

```
#import <Foundation/Foundation.h>
#import "MyNSBP_Desaturate.h"

// Our bundle is kept static so it can be accessed via class methods
static NSBundle* g_pluginBundle = nil;

@implementation MyNSBP_Desaturate

// Initialize
- (id)init
{
    self = [super init];
    return self;
}

// Deallocate
- (void)dealloc
{
    [super dealloc];
}

// Called to initialize our class, currently just to save our bundle
+ (BOOL)initializePluginClass: (NSBundle*)theBundle
{
    if (g_pluginBundle) {
        return NO;
    }
    g_pluginBundle = [theBundle retain];
    return YES;
}

// Called to terminate our class, currently just to release our bundle
+ (void)terminatePluginClass
{
    if (g_pluginBundle) {
        [g_pluginBundle release];
    }
}
```

```

        g_pluginBundle = nil;
    }
}

// Called to instantiate our plugin, currently to alloc, init and load the nib
+ (id) instantiatePlugin
{
    MyNSBP_Desaturate* instance =
        [[[MyNSBP_Desaturate alloc] init] autorelease];
    if (instance && [NSBundle loadNibNamed:@"Desaturate" owner:instance]) {
        return instance;
    }
    return nil;
}

// Do anything here to create UI, etc.
- (void) awakeFromNib
{
}

// Called to alter the image
- (NSImage*) doPlugin:(NSImage*) theImage
{
    NSRect theRect;

    // Calculate theRect of the image
    theRect.origin.x = theRect.origin.y = 0;
    theRect.size = [theImage size];

    // Lock the focus, draw, and unlock
    [theImage lockFocus];
    [[[NSColor lightGrayColor] colorWithAlphaComponent:0.1] set];
    NSRectFillUsingOperation(theRect, NSCompositeSourceOver);
    [theImage unlockFocus];

    // Return the image to the caller
    return theImage;
}

@end

```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 4—MyNSBP_RemoveColor

MyNSBP_RemoveColor.h

```
#import <Foundation/Foundation.h>
#import "MyNSBP_Protocol.h"

@interface MyNSBP_RemoveColor : NSObject<MyNSBP_Protocol> {
}
- (int)askUserWhatColorToRemove;
@end
```

MyNSBP_RemoveColor.m

```
#import <Foundation/Foundation.h>
#import "MyNSBP_RemoveColor.h"
#import "SettingsController.h"

// Our bundle is kept static so it can be accessed via class methods
static NSBundle* g_pluginBundle = nil;

@implementation MyNSBP_RemoveColor

// Initialize
- (id)init
{
    self = [super init];
    return self;
}

// Deallocate
- (void)dealloc
{
    [super dealloc];
}

// Called to initialize our class, currently just to save our bundle
+ (BOOL)initializePluginClass:(NSBundle*)theBundle
{
    if (g_pluginBundle) {
        return NO;
    }
    g_pluginBundle = [theBundle retain];
    return YES;
}

// Called to terminate our class, currently just to release our bundle
+ (void)terminatePluginClass
{
    if (g_pluginBundle) {
```

```

        [g_pluginBundle release];
        g_pluginBundle = nil;
    }
}

// Called to instantiate our plugin, currently to alloc, init and load the nib
+ (id)instantiatePlugin
{
    MyNSBP_RemoveColor* instance =
        [[[MyNSBP_RemoveColor alloc] init] autorelease];
    if (instance && [NSBundle loadNibNamed:@"RemoveColor" owner:instance]) {
        return instance;
    }
    return nil;
}

// Do anything here to create UI, etc.
- (void)awakeFromNib
{
}

// Ask the user what color to remove
- (int)askUserWhatColorToRemove
{
    int whichColor = 0;

    // Allocate our settings dialog box
    SettingsController *sc = [[SettingsController alloc] init];

    if (sc) {

        // Run the dialog box modal. The SettingsController will call
        // stopModalWithCode with the tag of the button that was pressed
        // to end the modal loop.
        [sc showWindow:self];
        whichColor = [NSApp runModalForWindow:[sc window]];

        // Deallocate the preference controller, we no longer need it
        [sc release];
        sc = nil;

    }

    // return the chosen color, or 0 (none)
    return whichColor;
}

// Called to alter the image
- (NSImage*)doPlugin:(NSImage*)theImage
{
    int whichColor = [self askUserWhatColorToRemove];

    // If the user chose a color (other than None)
    if (whichColor) {
        NSRect          theRect;

```

```

// Calculate theRect of the image
theRect.origin.x = theRect.origin.y = 0;
theRect.size = [theImage size];

// Lock the focus, draw, and unlock
[theImage lockFocus];

// Set the proper color
if (whichColor == 1) { // red
    [[NSColor redColor] set];
} else if (whichColor == 2) { // green
    [[NSColor greenColor] set];
} else if (whichColor == 3) { // blue
    [[NSColor blueColor] set];
}

// Fill the rect with the NSCompositePlusDarker mode to remove the color
NSRectFillUsingOperation(theRect, NSCompositePlusDarker);

// Unlock the focus
[theImage unlockFocus];
}

// Return the image to the caller (altered or not)
return theImage;
}

@end

```

SettingsController.h

```

#import <Cocoa/Cocoa.h>

@interface SettingsController : NSWindowController
{
}
- (IBAction)myDoneAction:(id) sender;
@end

```

SettingsController.m

```

#import "SettingsController.h"

@implementation SettingsController

- (id)init
{
    // Load the Settings.nib file
    self = [super initWithWindowNibName:@"Settings"];
    return self;
}

// Do stuff here if we need to once the window has loaded
- (void)windowDidLoad

```

```
{
}

- (IBAction)myDoneAction: (id) sender
{
    [NSApp stopModalWithCode: [sender tag]];
}

@end
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 4—plistPlugin

AppController.h

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

@interface AppController : NSObject
{
}
@end
```

AppController.m

```
#import "AppController.h"

@implementation AppController

- (id)init
{
    [super init];
    if (self) {

    }
    return self;
}

- (void)dealloc
{
    [super dealloc];
}

// Causes the application to quit when the one and only window is closed
- (BOOL)applicationShouldTerminateAfterLastWindowClosed: (NSApplication *) sender
{
    return TRUE;
}

@end
```

main.m

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

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

SearchWindowController.h

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

```
@interface SearchWindowController : NSObject  
{  
    IBOutlet NSProgressIndicator *_m_progressIndicator;  
    IBOutlet NSTextView *_m_resultsTextView;  
    IBOutlet NSButton *_m_searchButton;  
    IBOutlet NSPopUpButton *_m_searchEnginePopUpButton;  
    IBOutlet NSTextField *_m_searchTextField;  
    NSMutableArray *_m_searchEngineArray;  
}  
- (IBAction)doSearch: (id)sender;  
@end
```

```
SearchWindowController.m
```

```
#import "SearchWindowController.h"
```

```
@implementation SearchWindowController
```

```
- (id)init  
{  
    [super init];  
    if (self) {  
        m_searchEngineArray = [[NSMutableArray alloc] init];  
    }  
    return self;  
}  
  
- (void)dealloc  
{  
    [m_searchEngineArray release];  
    [super dealloc];  
}
```

```
// Called to initialize our fields if we need to -  
// this is where we will load our plugins
```

```
- (void)awakeFromNib  
{
```

```
    NSString* folderPath;
```

```
    // Remove all items from the popup button  
    [m_searchEnginePopUpButton removeAllItems];
```

```
    // Locate the plugins directory within our app package  
    folderPath = [[NSBundle mainBundle] builtInPlugInsPath];
```

```
    if (folderPath) {  
        // Enumerate through each plugin  
        NSEnumerator* enumerator = [[NSBundle pathsForResourceOfType:@"plist"  
                                     inDirectory:folderPath] objectEnumerator];  
        NSString* pluginPath;  
        while ((pluginPath = [enumerator nextObject])) {  
            // Load the plist into a dictionary for each plugin  
            NSDictionary* dictionary = [NSDictionary  
                                         dictionaryWithContentsOfFile:pluginPath];
```



```

        theFullRange withString:results];
    } else {
        NSRunAlertPanel(@"Error", @"NSString
        stringWithContentsOfURL returned nil",
        @"OK", @"", @"");
    }

    // End user feedback
    [m_searchButton setEnabled:YES];
    [m_progressIndicator stopAnimation:self];
}

@end

```

Google.plist

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>SearchEngineName</key>
    <string>Google</string>
    <key>SearchEngineURL</key>
    <string>http://www.google.com/search?q=</string>
</dict>
</plist>

```

Overture.plist

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>SearchEngineName</key>
    <string>Overture</string>
    <key>SearchEngineURL</key>
    <string>http://www.overture.com/d/search/?Keywords=</string>
</dict>
</plist>

```

Yahoo.plist

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>SearchEngineName</key>
    <string>Yahoo</string>
    <key>SearchEngineURL</key>
    <string>http://google.yahoo.com/bin/query?p=</string>

```

</di ct>
</pl i st>

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 5—CFPlugin

main.c

```
#include <Carbon/Carbon.h>
#include <CoreFoundation/CoreFoundation.h>
#include <CoreFoundation/CFPlugin.h>
#include <CoreFoundation/CFPlugInCOM.h>
#include "../MyCFPluginInterface.h"

// The layout for an instance of MyType
typedef struct _MyType {
    MyInterfaceStruct*    _myInterface;
    CFUUIDRef             _factoryID;
    UInt32                _refCount;
} MyType;

//
// Forward declarations for all functions
//

// IUnknown functions
static HRESULT myQueryInterface(void *this, REFIID iid, LPVOID *ppv);
static ULONG myAddRef(void *this);
static ULONG myRelease(void *this);

// My interface functions
static void myPluginFunction(void *this, Boolean flag);

// Utility functions
static MyType *_allocMyType(CFUUIDRef factoryID);
static void _deallocMyType(MyType *this);

// My factory function
void *myFactoryFunction(CFAllocatorRef allocator, CFUUIDRef typeId);

//
// IUnknown interface functions
//

// Implementation of the IUnknown QueryInterface function
static HRESULT myQueryInterface(void *this, REFIID iid, LPVOID *ppv)
{
    HRESULT hResult = E_NOINTERFACE;

    printf("myQueryInterface\n");

    // Create a CoreFoundation UUIDRef for the requested interface
    CFUUIDRef interfaceID = CFUUIDCreateFromUUIDBytes(NULL, iid);

    // Test the requested ID against the valid interfaces
    if (CFEqual(interfaceID, kMyInterfaceID)) {
```

```

// If the MyInterface was requested, bump the ref count,
// set the ppv parameter equal to the instance, and return good status
// This calls through to myAddRef
((MyType*)this)->_myInterface->AddRef(this);
*ppv = this;
hResult = S_OK;

```

```

} else if (CFEqual(interfaceID, IUnknownUUID)) {

```

```

// If the IUnknown interface was requested, bump the ref count,
// set the ppv parameter equal to the instance, and return good status
// This calls through to myAddRef
((MyType*)this)->_myInterface->AddRef(this);
*ppv = this;
hResult = S_OK;

```

```

} else {

```

```

// Requested interface unknown, bail with error
*ppv = NULL;
hResult = E_NOINTERFACE;

```

```

}

```

```

// Release interface
CFRelease(interfaceID);
return hResult;

```

```

}

```

```

// Implementation of reference counting for this type
// Whenever an interface is requested, bump the refCount for
// the instance NOTE: returning the refcount is a convention
// but is not required so don't rely on it

```

```

static ULONG myAddRef(void *this)
{
    printf("myAddRef\n");

    return ((MyType*)this)->_refCount++;
}

```

```

// When an interface is released, decrement the refCount
// If the refCount goes to zero, deallocate the instance
static ULONG myRelease(void *this)

```

```

{
    printf("myRelease\n");

    ((MyType*)this)->_refCount--;
    if (((MyType*)this)->_refCount == 0) {
        _deallocMyType((MyType*)this);
        return 0;
    } else
        return ((MyType*)this)->_refCount;
}

```

```

//
// Functions specific to my plugin
//

// The implementation of the MyInterface function
static void myPluginFunction(void *this, Boolean flag)
{
    SInt16 outItemHit;
    StandardAlert(kAlertNoteAlert,
        flag ? "\pCFPlugin (flag = YES)" : "\pCFPlugin (flag = NO)",
        "\pThis alert is being called from myPluginFunction in CFPlugin.",
        NULL, &outItemHit);
}

//
// Static definition of the MyInterface function table
//

// The MyInterface function table
static MyInterfaceStruct myInterfaceFtbl = {
    NULL, // Required padding for COM
    myQueryInterface, // These three are the required COM functions
    myAddRef,
    myRelease,
    myPluginFunction }; // Interface implementation (specific to my plugin)

//
// Utility functions for allocation and deallocation
//

// Utility function that allocates a new instance
static MyType *_allocMyType(CFUUIDRef factoryID)
{
    // Allocate memory for the new instance
    MyType *newOne = (MyType*) malloc(sizeof(MyType));

    // Point to the function table
    newOne->_myInterface = &myInterfaceFtbl;

    // Retain and keep an open instance refcount for each factory
    newOne->_factoryID = CFRetain(factoryID);
    CFPluginAddInstanceForFactory(factoryID);

    // This function returns the IUnknown interface so set the refCount to one
    newOne->_refCount = 1;
    return newOne;
}

// Utility function that deallocates the instance
// when the refCount goes to zero
static void _deallocMyType(MyType *this)
{
    CFUIDRef factoryID = this->_factoryID;
    free(this);
}

```

```
    if (factoryID) {
        CFPlugInRemoveInstanceForFactory(factoryID);
        CFRelease(factoryID);
    }
}

//
// Factory function
//

// Implementation of the factory function for this type
void *myFactoryFunction(CFAllocatorRef allocator, CFUUIDRef typeId)
{
    printf("myFactoryFunction\n");

    // If correct type is being requested,
    // allocate an instance of MyType and return the IUnknown interface
    if (CFEqual(typeID, kMyTypeID)) {
        MyType *result = _allocMyType(kMyFactoryID);
        return result;
    } else {
        // If the requested type is incorrect, return NULL
        return NULL;
    }
}
```

[\[Team LIB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 5—CFPluginCarbonApp

main.c

```
#include <Carbon/Carbon.h>
#include <CoreFoundation/CoreFoundation.h>
#include <CoreFoundation/CFPlugin.h>
#include <CoreFoundation/CFPluginCOM.h>
#include "../MyCFPluginInterface.h"
#include "../MyCFCallPlugin.h"

int main(int argc, char* argv[])
{
    I BNibRef          nibRef;
    WindowRef         window;
    OSStatus          err;

    // Create a Nib reference passing the name of the nib file
    // (without the .nib extension)
    // CreateNibReference only searches in the application bundle.
    err = CreateNibReference(CFSTR("main"), &nibRef);
    require_noerr(err, CantGetNibRef);

    // Once the nib reference is created, set the menu bar.
    // "MainMenu" is the name of the menu bar object.
    // This name is set in InterfaceBuilder when the nib is created.
    err = SetMenuBarFromNib(nibRef, CFSTR("MenuBar"));
    require_noerr(err, CantSetMenuBar);

    // Then create a window.
    // "MainWindow" is the name of the window object.
    // This name is set in InterfaceBuilder when the nib is created.
    err = CreateWindowFromNib(nibRef, CFSTR("MainWindow"), &window);
    require_noerr(err, CantCreateWindow);

    // We don't need the nib reference anymore.
    DisposeNibReference(nibRef);

    // The window was created hidden so show it.
    ShowWindow(window);

    // Call the plugin
    CallPlugin();

    // Call the event loop
    RunApplicationEventLoop();

CantCreateWindow:
CantSetMenuBar:
CantGetNibRef:
    return err;
}
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 5—CFPluginCocoaApp

AppController.h

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

@interface AppController : NSObject
{
}
- (IBAction)doCallPlugin: (id) sender;
@end
```

AppController.m

```
#import "AppController.h"
#import "../MyCFPluginInterface.h"
#import "../MyCFCallPlugin.h"

@implementation AppController

- (id)init
{
    [super init];
    if (self) {
    }
    return self;
}

- (void)dealloc
{
    [super dealloc];
}

// Causes the application to quit when the one and only window is closed
- (BOOL)applicationShouldTerminateAfterLastWindowClosed: (NSApplication *) sender
{
    return TRUE;
}

- (void)awakeFromNib
{
    // CallPlugin();
}

- (IBAction)doCallPlugin: (id) sender
{
    CallPlugin();
}

@end
```

main.m

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

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

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 5—Shared

MyCFCallPlugin.c

```
#include <Carbon/Carbon.h>
#include <CoreFoundation/CoreFoundation.h>
#include <CoreFoundation/CFPlugin.h>
#include <CoreFoundation/CFPluginCOM.h>
#include "MyCFPluginInterface.h"
#include "MyCFCallPlugin.h"

void CallPlugin(void)
{
    // Create a URL that points to the plug-in using a hard-coded path
    // (You will need to change this to point to the plugin)
    CFURLRef url = CFURLCreateWithFilePath(NULL,
        CFSTR("/Users/zobkiw/Documents/OSXBook/_SOURCE_/5. Carbon
            Plugins/CFPlugin/bui ld/CFPlugin.plugin"),
        kCFURLPOSIXPathStyle, TRUE);

    // Create a CFPlugin using the URL
    // This step causes the plug-in's types and
    // factories to be registered with the system
    // Note that the plug-in's code is not loaded
    // unless the plug-in is using dynamic registration
    CFPluginRef plugin = CFPluginCreate(NULL, url);
    if (plugin) {
        // See if this plug-in implements the "My" type
        CFArrayRef factories = CFPluginFindFactoriesForPluginType(kMyTypeID);

        // If there are factories for the requested type,
        // attempt to get the IUnknown interface
        if ((factories != NULL) && (CFArrayGetCount(factories) > 0)) {

            // Get the factory ID for the first location in the array of IDs
            CFUUIDRef factoryID = CFArrayGetValueAtIndex(factories, 0);

            // Use the factory ID to get an IUnknown interface
            // Here the code for the PlugIn is loaded
            // IUnknownVtbl is a struct containing the IUNKNOWN_C_GUTS
            // CFPlugin::MyFactoryFunction is called here
            IUnknownVtbl **iunknown =
                CFPluginInstanceCreate(NULL, factoryID, kMyTypeID);

            // If this is an IUnknown interface, query for the "My" interface
            if (iunknown) {
                MyInterfaceStruct **interface = NULL;

                // CFPlugin::myQueryInterface is called here
                // CFPlugin::myAddRef is called here
                (*iunknown)->QueryInterface(iunknown,
```

```
CFUIDGetUIDBytes(kMyInterfaceID),
(LPVOID *)(&interface));
```

```
// Done with IUnknown
// CFPlugin::myRelease is called here
(*iunknown) ->Release(iunknown);
```

```
// If this is a "My" interface, try to call its function
if (interface) {
    (*interface) ->myPluginFunction(interface, TRUE);
    (*interface) ->myPluginFunction(interface, FALSE);
```

```
    // Done with interface
    // This causes the plug-in's code to be unloaded
    // CFPlugin::myRelease is called here
    (*interface) ->Release(interface);
```

```
    } else printf( "Failed to get interface.\n" );
```

```
    } else printf( "Failed to create instance.\n" );
```

```
} else printf( "Could not find any factories.\n" );
```

```
// Release the CFPlugin memory
```

```
CFRelease(plugin);
```

```
} else printf( "Could not create CFPluginRef.\n" );
```

```
}
```

MyCFCallPlugin.h

```
void CallPlugin(void);
```

MyCFPluginInterface.h

```
/*
TYPE
DC83B5C8- DD18- 11D6- B3D0- 0003930EDB36
0xDC, 0x83, 0xB5, 0xC8, 0xDD, 0x18, 0x11, 0xD6,
0xB3, 0xD0, 0x00, 0x03, 0x93, 0x0E, 0xDB, 0x36
```

```
FACTORY
DC3F0D89- DD19- 11D6- B3D0- 0003930EDB36
0xDC, 0x3F, 0x0D, 0x89, 0xDD, 0x19, 0x11, 0xD6,
0xB3, 0xD0, 0x00, 0x03, 0x93, 0x0E, 0xDB, 0x36
```

```
INTERFACE
071D881D- DD1A- 11D6- B3D0- 0003930EDB36
0x07, 0x1D, 0x88, 0x1D, 0xDD, 0x1A, 0x11, 0xD6,
0xB3, 0xD0, 0x00, 0x03, 0x93, 0x0E, 0xDB, 0x36
*/
```

```
//#include <Carbon/Carbon.h>
#include <CoreFoundation/CoreFoundation.h>
#include <CoreFoundation/CFPlugin.h>
#include <CoreFoundation/CFPluginCOM.h>
```

```
// Define the UUID for the type
#define kMyTypeID (CFUUIDGetConstantUUIDWithBytes(
NULL, 0xDC, 0x83, 0xB5, 0xC8, 0xDD, 0x18,
0x11, 0xD6, 0xB3, 0xD0, 0x00, 0x03, 0x93, 0x0E, 0xDB, 0x36))

// The UUID for the factory function
#define kMyFactoryID (CFUUIDGetConstantUUIDWithBytes(
NULL, 0xDC, 0x3F, 0x0D, 0x89, 0xDD, 0x19,
0x11, 0xD6, 0xB3, 0xD0, 0x00, 0x03, 0x93, 0x0E, 0xDB, 0x36))

// Define the UUID for the interface
// MyType objects must implement MyInterface
#define kMyInterfaceID (CFUUIDGetConstantUUIDWithBytes(
NULL, 0x07, 0x1D, 0x88, 0x1D, 0xDD, 0x1A,
0x11, 0xD6, 0xB3, 0xD0, 0x00, 0x03, 0x93, 0x0E, 0xDB, 0x36))

// The function table for the interface
typedef struct MyInterfaceStruct {
    IUNKNOWN_C_GUTS;
    void (*myPluginFunction)(void *this, Boolean flag);
} MyInterfaceStruct;
```

[\[Team LIB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 6—MyCarbonFramework

MyCarbonFramework.c

```
#include "MyCarbonFramework.h"

int doAddition(int a, int b)
{
    return a+b;
}
```

MyCarbonFramework.h

```
#include <Carbon/Carbon.h>

int doAddition(int a, int b);
```

Chapter 6—MyCocoaFramework

MyCocoaFramework.h

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

@interface MyCocoaObject : NSObject
{
}
+ (int)doAddition: (int) a plus: (int) b;
@end
```

MyCocoaFramework.m

```
#import "MyCocoaFramework.h"

@implementation MyCocoaObject

+ (int)doAddition: (int) a plus: (int) b
{
    return a+b;
}

@end
```

Chapter 6—MyCocoaApp

AppController.h

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

@interface AppController : NSObject
{
    IBOutlet NSTextField *m_a;
    IBOutlet NSTextField *m_b;
}
- (IBAction)doCarbon: (id) sender;
- (IBAction)doCocoa: (id) sender;
@end
```

AppController.m

```
#import "AppController.h"
#import "../MyCocoaFramework/MyCocoaFramework.h"
#import "../MyCarbonFramework/MyCarbonFramework.h"

@implementation AppController

- (void)awakeFromNib
{
    [m_a setIntValue: 7];
    [m_b setIntValue: 3];
}

- (IBAction)doCarbon: (id) sender
{
    if (NSRunAlertPanel(@"Carbon", @"Call the Carbon framework?",
        @"OK", @"Cancel", @"") == NSAlertDefaultReturn) {
        int result = doAddition([m_a intValue], [m_b intValue]);
        NSRunAlertPanel(@"Carbon Result",
            @"The result of the computation is %d",
            @"OK", @"", @"", result);
    }
}

- (IBAction)doCocoa: (id) sender
{
    if (NSRunAlertPanel(@"Cocoa", @"Call the Cocoa framework?",
        @"OK", @"Cancel", @"") == NSAlertDefaultReturn) {
        int result = [MyCocoaObject doAddition:
            [m_a intValue] plus: [m_b intValue]];
        NSRunAlertPanel(@"Cocoa Result",
            @"The result of the computation is %d",
            @"OK", @"", @"", result);
    }
}
}
```

```
// Causes the application to quit when the one and only window is closed
- (BOOL) applicationShouldTerminateAfterLastWindowClosed: (NSApplication *) sender
{
    return TRUE;
}
```

@end

main.m

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

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

[\[Team Lib \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 7—MyTextService

main.m

```
#import <Foundation/Foundation.h>
#import "MyTextService.h"

int main(int argc, const char *argv[]) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    // Allocate and initialize our service provider
    MyTextService *serviceProvider = [[MyTextService alloc] init];

    // Register the service provider as such
    NSRegisterServicesProvider(serviceProvider, @"MyTextService");

    NS_DURING
        // Configure this application to run as a server
        [[NSRunLoop currentRunLoop] configureAsServer];

        // Run the application, use runUntilDate to make your
        // application auto-quit
        [[NSRunLoop currentRunLoop] run];
    NS_HANDLER
        NSLog(@"%@", [localException]);
    NS_ENDHANDLER

    // Release the service provider
    [serviceProvider release];

    [pool release];

    exit(0);          // insure the process exit status is 0
    return 0;        // ...and make main fit the ANSI spec.
}
```

MyTextService.h

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

@interface MyTextService : NSObject {
}

- (void)changeCase:(NSPasteboard *)pboard
                userData:(NSString *)userData
                error:(NSString **)error;

@end
```

MyTextService.m

```

#import "MyTextService.h"

@implementation MyTextService

- (void)changeCase: (NSPasteboard *)pboard
    userData: (NSString *)userData
    error: (NSString **)error
{
    NSString *pboardString;
    NSString *newString;
    NSArray *types;
    Boolean success;

    // Verify the types currently on the pasteboard
    types = [pboard types];
    if (![types containsObject: NSStringPboardType]) {
        *error = NSLocalizedString(@"Error: couldn't edit text.",
            @"pboard doesn't have a string.");
        return;
    }
    pboardString = [pboard stringForType: NSStringPboardType];
    if (!pboardString) {
        *error = NSLocalizedString(@"Error: couldn't edit text.",
            @"pboard couldn't give string.");
        return;
    }

    // Compare the mode so we do the correct thing
    if ([userData isEqualToString:@"upper"]) {
        newString = [pboardString uppercaseString];
    } else if ([userData isEqualToString:@"lower"]) {
        newString = [pboardString lowercaseString];
    } else if ([userData isEqualToString:@"cap"]) {
        newString = [pboardString capitalizedString];
    }
    if (!newString) {
        *error = NSLocalizedString(@"Error: couldn't edit text.",
            @"pboardString couldn't edit letters.");
        return;
    }
    types = [NSArray arrayWithObject: NSStringPboardType];

    // Load the value of the checkbox from the Preference Pane example
    // Force a synchronization so we get the latest "live" preference
    success = CFPreferencesAppSynchronize
        (CFSTR("com.triplesoft.mypreferencepane"));
    if (success && CFPreferencesGetAppBooleanValue
        (CFSTR("Boolean Value Key"),
            CFSTR("com.triplesoft.mypreferencepane"), NULL)) {
        // Use [NSPasteboard generalPasteboard] instead of pboard
        // to send the results to the clipboard
        [[NSPasteboard generalPasteboard] declareTypes: types owner: nil];
        [[NSPasteboard generalPasteboard] setString: newString
            forType: NSStringPboardType];
    } else {

```

```
// Send the results directly back to the app who
// requested it if set or if key does not exist
[pcb declareTypes: types owner: nil];
[pcb setString: newString ofType: NSStringPboardType];
}

return;
}

@end
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 8—MyPreferencePane

MyPreferencePanePref.h

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

@interface MyPreferencePanePref : NSPreferencePane
{
    CFStringRef m_appID;           // application ID (this application)

    // Tab 1
    IBOutlet NSButton *m_checkBox;
    IBOutlet NSTextField *m_textField;

    // Tab 2
    IBOutlet NSButton *m_pushButton;
}

- (IBAction)checkboxClicked: (id) sender;
- (IBAction)buttonClicked: (id) sender;

@end
```

MyPreferencePanePref.m

```
#import "MyPreferencePanePref.h"

@implementation MyPreferencePanePref

// This function is called as we are being initialized
- (id) initWithBundle: (NSBundle *) bundle
{
    // Initialize the location of our preferences
    if ((self = [super initWithBundle: bundle]) != nil) {
        m_appID = CFSTR("com.triplesoft.mypreferencepane");
    }

    return self;
}

// This function is called once the Nib is loaded
// and our windows are ready to be initialized.
- (void)mainViewDidLoad
{
    CFPropertyListRef value;

    // Load the value of the checkbox as a BOOLEAN
    value = CFPreferencesCopyAppValue(CFSTR("Boolean Value Key"), m_appID);
```

```

if (value && CFGetTypeID(value) == CFBooleanGetTypeID()) {
    [m_checkBox setState:CFBooleanGetValue(value)];
} else {
    [m_checkBox setState:NO];    // Default value of the checkbox
}
if (value) CFRelease(value);

// Load the value of the text field as a STRING
value = CFPreferencesCopyAppValue(CFSTR("String Value Key"), m_appID);
if (value && CFGetTypeID(value) == CFStringGetTypeID()) {
    [m_textField setStringValue:(NSString *)value];
} else {
    [m_textField setStringValue:@""];    // Default value of the text field
}
if (value) CFRelease(value);
}

```

```

// This action is called when our checkbox is
// clicked, we save the value immediately
- (IBAction)checkboxClicked:(id)sender
{
    CFPreferencesSetAppValue(CFSTR("Boolean Value Key"),
        [sender state] ? kCFBooleanTrue : kCFBooleanFalse, m_appID);

    // Force a synchronization so our preference is "live"
    CFPreferencesAppSynchronize(m_appID);
}

```

```

// This function is called when our preference
// pane is being deselected. That is, either
// another preference pane is being selected or
// the System Preferences application is
// shutting down. In this case we will save our text
// field, which is not saved at any other
// time. The checkbox is saved as it is clicked but
// just as easily can be saved here.
- (void)didUnselect
{
    CFNotificationCenterRef center;

    // Save text field
    CFPreferencesSetAppValue(CFSTR("String Value Key"),
        [m_textField stringValue], m_appID);

    // Write out all the changes that have been made for this application
    CFPreferencesAppSynchronize(m_appID);

    // Post a notification that the preferences
    // for this application have changed,
    // any observers will then become the first to know that this has occurred.
    center = CFNotificationCenterGetDistributedCenter();
    CFNotificationCenterPostNotification(center, CFSTR("Preferences Changed"),
        m_appID, NULL, TRUE);
}

```

```

// This action is called when the button on the second tab is clicked

```

```
- (IBAction)buttonClicked: (id) sender
{
    NSBeep();
}

@end
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 9—MyStatusItem

AppController.h

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

@interface AppController : NSObject
{
    NSStatusItem      *m_item;
    NSMenu             *m_menu;
}

- (void) selItemOne: (id) sender;
- (void) selItemTwo: (id) sender;
- (void) selItemIP: (id) sender;
- (void) selItemQuit: (id) sender;

@end
```

AppController.m

```
#import "AppController.h"

@implementation AppController

- (id) init
{
    [super init];
    if (self) {
        // Get the system status bar
        NSStatusBar *bar = [NSStatusBar systemStatusBar];

        // #define USE_TEXT_TITLE 1
        // Add an item to the status bar
        #ifdef USE_TEXT_TITLE
            m_item = [[bar statusItemWithLength:
                NSVariableStatusItemLength] retain];
            [m_item setTitle: NSLocalizedString(@"My", @"")];
        #else
            m_item = [[bar statusItemWithLength: NSSquareStatusItemLength] retain];
            [m_item setImage: [NSImage imageNamed: @"x"]];
        #endif

        [m_item setHighlightMode: YES];

        // Create a menu to be added to the item
        m_menu = [[[NSMenu alloc] init] retain];
        [m_menu setAutoenablesItems: YES];
        [m_menu addItem: [[[NSMenuItem alloc] initWithTitle:
            NSLocalizedString(@"Item One", @"")
            action: @selector(selItemOne:) keyEquivalent: @""]]];
    }
}
```

```

[m_menu addItem: [[NSMenuItem alloc] initWithTitle:
                  NSLocalizedString(@"Item Two", @"")
                  action:@selector(selItemTwo:) keyEquivalent:@""]];
[m_menu addItem: [NSMenuItem separatorItem]];
[m_menu addItem: [[NSMenuItem alloc] initWithTitle:
                  NSLocalizedString(@"My IP", @"")
                  action:@selector(selItemIP:) keyEquivalent:@""]];
[m_menu addItem: [NSMenuItem separatorItem]];
[m_menu addItem: [[NSMenuItem alloc] initWithTitle:
                  NSLocalizedString(@"Quit MyStatusItem", @"")
                  action:@selector(selItemQuit:) keyEquivalent:@""]];

```

```

// And add the menu to the item
[m_item setMenu:m_menu];

```

```

}
return self;
}

```

```

- (void) dealloc
{

```

```

    [m_item release];
    [m_menu release];
    [super dealloc];
}

```

```

- (BOOL) validateMenuItem: (NSMenuItem *) menuItem
{

```

```

    NSString *selectorString;
    selectorString = NSStringFromSelector([menuItem action]);

```

```

    NSLog(@"validateMenuItem called for %@", selectorString);
    if ([menuItem action] == @selector(selItemOne:))
        return YES;
    if ([menuItem action] == @selector(selItemTwo:))
        return YES;
    if ([menuItem action] == @selector(selItemIP:))
        return YES;
    if ([menuItem action] == @selector(selItemQuit:))
        return YES;

```

```

    return NO;
}

```

```

- (void) selItemOne: (id) sender
{

```

```

    int result;
    NSBeep();
    result = NSRunAlertPanel(@"MyStatusItem",
                            @"Thank you for selecting menu item one.",
                            @"OK", @"", @"");
}

```

```

- (void) selItemTwo: (id) sender
{

```

```

    int result;

```

```

NSBeep();
result = NSRunAlertPanel(@"MyStatusItem",
    @"Thank you for selecting menu item two.",
    @"OK", @"", @"");
}

- (void) selItemIP: (id) sender
{
    int result;
    NSString *theString = [NSString
        localizedStringWithFormat:@"IP Address: %@\nDNS Name: %@",
        [[NSHost currentHost] address], [[NSHost currentHost] name]];
    NSBeep();
    result = NSRunAlertPanel(@"MyStatusItem", theString, @"OK", @"", @"");
}

- (void) selItemQuit: (id) sender
{
    int result;
    NSBeep();
    result = NSRunAlertPanel(@"MyStatusItem",
        @"Quit the application and remove this menu?", @"Yes", @"No", @"");
    if (result == NSAlertDefaultReturn) {
        [[NSApplication sharedApplication] terminate:self];
        // Respond to applicationWillTerminate for cleanup
    }
}

@end

main.m

#import <Cocoa/Cocoa.h>

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

```

[\[Team LiB \]](#)

◀ PREVIOUS NEXT ▶

Chapter 10—MyScreenEffect

MyScreenSaverView.h

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

@interface MyScreenSaverView : ScreenSaverView
{
    int m_version;           // Used to tell the version of our
                           // preferences (and if they loaded)
    int m_useTransparency; // Use transparency when drawing the stars
    int m_useColor;        // Use color when drawing the stars

    IBOutlet id            m_configureSheet;
    IBOutlet NSButton     *m_useTransparencyCheckbox;
    IBOutlet NSButton     *m_useColorCheckbox;
}

- (IBAction) closeSheet_save:(id) sender;
- (IBAction) closeSheet_cancel:(id) sender;

@end
```

MyScreenSaverView.m

```
#import "MyScreenSaverView.h"

// Define what defines our bundle, this can be done a variety of ways include
// creating a global NSString
#define kBundleID @"com.triplesoft.myscreensaver"

@implementation MyScreenSaverView

- (id)initWithFrame:(NSRect) frame isPreview:(BOOL) isPreview
{
    // Initialize our super - the ScreenSaverView
    self = [super initWithFrame:frame isPreview:isPreview];

    if (self) { // If all went well...

        // Load the defaults for this screensaver
        ScreenSaverDefaults *defaults = [ScreenSaverDefaults
            defaultsForModuleWithName:kBundleID];

        // Try to load the version information to
        // see if we have any saved settings
        m_version = [defaults floatForKey:@"version"];
        if (!m_version) {
            // No saved setting, define our defaults
            m_version = 1;
            m_useTransparency = YES;
        }
    }
}
```

```

    m_useColor = YES;

    // Now write out the defaults to the defaults database
    [defaults setInteger:m_version forKey:@"version"];
    [defaults setInteger:m_useTransparency forKey:@"useTransparency"];
    [defaults setInteger:m_useColor forKey:@"useColor"];

    // And synchronize
    [defaults synchronize];
}

// Now, although this is overkill if version == 0,
// we load the defaults from the
// defaults database into our member variables
m_useTransparency = [defaults integerForKey:@"useTransparency"];
m_useColor = [defaults integerForKey:@"useColor"];

// And set the animation time interval
// (how often animateOneFrame is called)
[self setAnimationTimeInterval:1/30.0];
}

```

```
return self;
```

```
}
```

```
- (void)startAnimation
```

```
{
    [super startAnimation];
}
```

```
- (void)stopAnimation
```

```
{
    [super stopAnimation];
}
```

```
- (void)drawRect:(NSRect)rect
```

```
{
    // We could draw here, but instead we draw in animateOneFrame
    [super drawRect:rect];
}
```

```
- (void)animateOneFrame
```

```
{
    NSRect    rect;
    NSColor   *color;
    int       x;
    float     r;
```

```
// Calculate the star size
x = SSRandomIntBetween(1, 3);
```

```
// Make a rectangle at a random location the size of the star
rect.origin = SSRandomPointForSizeWithinRect
    (NSMakeSize(x, x), [self bounds]);
rect.size = NSMakeSize(x, x);
```

```

// Calculate a random color (or gray), with or without transparency,
// depending on the user preference
r = SSRandomFloatBetween(0.0, 1.0);
color = [NSColor colorWithCalibratedRed:r // use red
        // use new color or same as red if not using color
        green:m_useColor ? SSRandomFloatBetween(0.0, 1.0) : r
        // use new color or same as red if not using color
        blue:m_useColor ? SSRandomFloatBetween(0.0, 1.0) : r
        // use transparency if the user says so
        alpha:m_useTransparency ?
        SSRandomFloatBetween(0.0, 1.0) : 1.0];
[color set];

```

```

// Draw the star
[[NSBezierPath bezierPathWithOvalInRect:rect] fill];

```

```

return;
}

```

```

- (BOOL)hasConfigureSheet
{

```

```

    return YES;
}

```

```

// Display the configuration sheet for the user to choose their settings
- (NSWindow*)configureSheet
{

```

```

    // If we have yet to load our configure sheet,
    // load the nib named MyScreenSaver.nib
    if (!m_configureSheet)
        [NSBundle loadNibNamed:@"MyScreenSaver" owner:self];

```

```

    // Set the state of our UI components
    [m_useTransparencyCheckbox setState:m_useTransparency];
    [m_useColorCheckbox setState:m_useColor];

```

```

    return m_configureSheet;
}

```

```

// The user clicked the SAVE button in the configuration sheet
- (IBAction)closeSheet_save:(id) sender
{

```

```

    // Get our defaults
    ScreenSaverDefaults *defaults =
        [ScreenSaverDefaults defaultsForModuleWithName:kBundleID];

```

```

    // Save the state of our UI components
    m_useTransparency = [m_useTransparencyCheckbox state];
    m_useColor = [m_useColorCheckbox state];

```

```

    // Write them to the defaults database
    [defaults setInteger:m_useTransparency forKey:@"useTransparency"];
    [defaults setInteger:m_useColor forKey:@"useColor"];

```

```

    // Synchronize
    [defaults synchronize];
}

```

```
// The sheet has ended, go in peace
[NSApp endSheet:m_configureSheet];
}

// The user clicked the CANCEL button in the configuration sheet
- (IBAction) closeSheet_cancel:(id) sender
{
    // Nothing to do! The sheet has ended, go in peace
    [NSApp endSheet:m_configureSheet];
}

@end
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 11—MyColorPicker

ThePicker.h

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

@interface ThePicker : NSColorPicker <NSColorPickingCustom>
{
    IBOutlet NSWindow *m_window;
    IBOutlet NSBox *m_box;
    IBOutlet NSTextView *m_textView;
    IBOutlet NSColorWell *m_colorWell;

    NSColor *m_color;
}

- (IBAction) doRandomColor: (id) sender;
- (void) logText: (NSString *) theString;

@end
```

ThePicker.m

```
#import "ThePicker.h"

#define ThePickerMode 100

@implementation ThePicker

// ----- NSColorPickingDefault

- (id) initWithPickerMask: (int) mask colorPanel: (NSColorPanel *) owningColorPanel
{
    // [self logText:@"initWithPickerMask\n"];
    if (mask & NSColorPanelRGBModeMask) { // we only support RGB mode
        [super initWithPickerMask: mask colorPanel: owningColorPanel];
    }
    srand(time(0)); // init random number seed
    return self;
}

- (NSImage *) provideNewButtonImage
{
    // [self logText:@"provideNewButtonImage\n"];
    return [[NSImage alloc] initWithContentsOfFile: [[NSBundle bundleForClass:
        [self class]] pathForResource:@"ThePicker"]];
}

// ----- NSColorPickingCustom

- (void) setColor: (NSColor *) color
```

```

{
    [self logText:@"setColor\n"];
    [color retain];
    [m_color release];
    m_color = color;
    [m_colorWell setColor:m_color];
}

- (int)currentMode
{
    [self logText:@"currentMode\n"];
    return ThePickerMode;
}

- (BOOL)supportsMode: (int) mode
{
    [self logText:@"supportsMode\n"];
    return (mode == ThePickerMode);
}

- (NSView *)provideNewView: (BOOL) initialRequest
{
    if (initialRequest) {
        if ([NSBundle loadNibNamed:@"ThePicker" owner:self]) {
            [self logText:@"provideNewView\n"];
            return m_box;
        } else {
            NSBeep();
            NSRunAlertPanel(@"Error", @"Couldn't load nib.", @"OK", @"", @"");
        }
    }
    return m_box;
}

// ----- Actions

static __inline__ float SSRandomFloatBetween(float a, float b)
{
    return a + (b - a) * ((float)random() / (float) LONG_MAX);
}

- (IBAction)doRandomColor: (id) sender
{
    NSColor *theColor;
    [self logText:@"doRandomColor\n"];
    theColor = [NSColor colorWithCalibratedRed: SSRandomFloatBetween(0.0, 1.0)
        green: SSRandomFloatBetween(0.0, 1.0)
        blue: SSRandomFloatBetween(0.0, 1.0)
        alpha: SSRandomFloatBetween(0.0, 1.0)];
    [[self colorPanel] setColor: theColor];
}

- (void)logText: (NSString *) theString
{
    // Append the text to the end of the text view and scroll it into view
    NSRange theEnd = NSMakeRange([[m_textView string] length], 0);

```

```
[m_textView replaceCharactersInRange: theEnd withString: theString];  
theEnd.location += [theString length];  
[m_textView scrollRangeToVisible: theEnd];  
}
```

@end

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 12—MyThread

AppController.h

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

@interface AppController : NSObject
{
    IBOutlet NSButton          *m_startButton;
    IBOutlet NSButton          *m_stopButton;
    IBOutlet NSProgressIndicator *m_indicator1;
    IBOutlet NSProgressIndicator *m_indicator2;
    IBOutlet NSProgressIndicator *m_indicator3;

    NSConditionLock             *m_conditionLock;
    BOOL                         m_stop;
}
- (IBAction) start: (id) sender;
- (IBAction) stop: (id) sender;

- (void) thread1: (id) owner;
- (void) thread2: (id) owner;
- (void) thread3: (id) owner;

- (void) cleanupThread: (id) owner;

@end
```

AppController.m

```
#import "AppController.h"

#define CONDITION_TASK_ONE      1
#define CONDITION_TASK_TWO      2
#define CONDITION_TASK_THREE    3
#define CONDITION_TASK_CLEANUP  4

@implementation AppController

// This action is called when the Start button is pressed, it begins everything
- (IBAction) start: (id) sender
{
    if (m_conditionLock == nil) {
        // Start out in Condition 1 so things start up right away
        m_conditionLock = [[NSConditionLock alloc]
            initWithCondition: CONDITION_TASK_ONE];

        // Set stop flag
        m_stop = NO;

        // Create all the threads that will eventually run based on Condition
```

```

    [NSThread detachNewThreadSelector:@selector(thread1:)
      toTarget:self withObject:self];
    [NSThread detachNewThreadSelector:@selector(thread2:)
      toTarget:self withObject:self];
    [NSThread detachNewThreadSelector:@selector(thread3:)
      toTarget:self withObject:self];

    // Create a final, cleanup thread
    [NSThread detachNewThreadSelector:@selector(cleanupThread:)
      toTarget:self withObject:self];

    [m_startButton setEnabled:NO];
    [m_stopButton setEnabled:YES];
} else {
    NSRunAlertPanel(@"Error", @"m_conditionLock is not nil",
      @"OK", @"", @"");
}
}

// This action is called when the Stop button is pressed,
// it sets a flag that the threads check
- (IBAction)stop:(id)sender
{
    [m_stopButton setEnabled:NO];
    m_stop = YES;
}

// This is the first thread to run, it starts as soon as
// the Start button is pressed
- (void)thread1:(id)owner
{
    int x;

    // Threads are responsible to manage their own autorelease pools
    NSAutoreleasePool *thePool = [[NSAutoreleasePool alloc] init];

    // Reset the progress indicator
    [m_indicator1 setDoubleValue:0];

    // Wait until Condition 1
    [m_conditionLock lockWhenCondition:CONDITION_TASK_ONE];

    // Loop, checking the stop flag
    for (x=1; x<=100; ++x) {
        if (m_stop) break;
        [m_indicator1 setDoubleValue:x];
        [NSThread sleepUntilDate:[NSDate dateWithTimeIntervalSinceNow:
          (NSTimeInterval)0.05]];
    }

    // Change to Condition 2
    [m_conditionLock unlockWhenCondition:CONDITION_TASK_TWO];

    // Release the autoreleasepool
    [thePool release];
}

```

```

    // Exit this thread
    [NSThread exit];
}

// This is the second thread to run, it starts as soon as the Start
// button is pressed, then waits until the first thread is finished
- (void)thread2: (id)owner
{
    int x;

    // Threads are responsible to manage their own autorelease pools
    NSAutoreleasePool *thePool = [[NSAutoreleasePool alloc] init];

    // Reset the progress indicator
    [m_indicator2 setDoubleValue: 0];

    // Wait until Condition 2
    [m_conditionLock lockWhenCondition: CONDITION_TASK_TWO];

    // Loop, checking the stop flag
    for (x=1; x<=100; ++x) {
        if (m_stop) break;
        [m_indicator2 setDoubleValue: x];
        [NSThread sleepUntilDate: [NSDate dateWithTimeIntervalSinceNow:
            (NSTimeInterval) 0.05]];
    }

    // Change to Condition 3
    [m_conditionLock unlockWithCondition: CONDITION_TASK_THREE];

    // Release the autoreleasepool
    [thePool release];

    // Exit this thread
    [NSThread exit];
}

// This is the third thread to run, it starts as soon as the Start
// button is pressed, then waits until the second thread is finished
- (void)thread3: (id)owner
{
    int x;

    // Threads are responsible to manage their own autorelease pools
    NSAutoreleasePool *thePool = [[NSAutoreleasePool alloc] init];

    // Reset the progress indicator
    [m_indicator3 setDoubleValue: 0];

    // Wait until Condition 3
    [m_conditionLock lockWhenCondition: CONDITION_TASK_THREE];

    // Loop, checking the stop flag
    for (x=1; x<=100; ++x) {
        if (m_stop) break;
        [m_indicator3 setDoubleValue: x];
    }
}

```

```

        [NSThread sleepUntilDate: [NSDate dateWithTimeIntervalSinceNow:
            (NSTimeInterval) 0.05]];
    }

    // Change to Condition 4
    [m_conditionLock unlockWithCondition: CONDITION_TASK_CLEANUP];

    // Release the autoreleasepool
    [thePool release];

    // Exit this thread
    [NSThread exit];
}

// This is the cleanup thread, it starts as soon as the Start
// button is pressed, then waits until the third thread is finished
- (void) cleanupThread: (id) owner
{
    // Threads are responsible for manage their own autorelease pools
    NSAutoreleasePool *thePool = [[NSAutoreleasePool alloc] init];

    // Wait until Condition 4
    [m_conditionLock lockWhenCondition: CONDITION_TASK_CLEANUP];

    // Last stop, clean up
    [m_conditionLock unlock];
    [m_conditionLock release];
    m_conditionLock = nil;

    // Update the UI
    [m_stopButton setEnabled: NO];
    [m_startButton setEnabled: YES];

    // Play done sound if the user didn't stop prematurely
    if (!m_stop)
        [[NSSound soundNamed:@"done"] play];

    // Release the autoreleasepool
    [thePool release];

    // Exit this thread
    [NSThread exit];
}

// Causes the application to quit when the one and only window is closed
- (BOOL) applicationShouldTerminateAfterLastWindowClosed: (NSApplication *) sender
{
    return TRUE;
}

- (void) windowWillClose: (NSNotification *) aNotification
{
    [self stop:self];
}

@end

```

main.m

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

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

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 13—MyTerminal

AppController.h

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

@interface AppController : NSObject
{
}
@end
```

AppController.m

```
#import "AppController.h"

@implementation AppController

- (id)init
{
    [super init];
    if (self) {

    }
    return self;
}

- (void)dealloc
{
    [super dealloc];
}

// Causes the application to quit when the one and only window is closed
- (BOOL)applicationShouldTerminateAfterLastWindowClosed: (NSApplication *) sender
{
    return TRUE;
}

@end
```

main.m

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

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

MyTerminalController.h

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

```
@interface MyTerminalController : NSObject
```

```
{
    IBOutlet NSTextView          *m_textView;
    IBOutlet NSButton           *m_pingButton;
    IBOutlet NSProgressIndicator *m_pingIndicator;

    IBOutlet NSTextField        *m_textField;
    IBOutlet NSButton           *m_uptimeButton;

    NSTask                      *m_pingTask;
    NSPipe                       *m_pingPipe;
    BOOL                         m_pingIsRunning;
}
- (IBAction) ping: (id) sender;
- (IBAction) uptime: (id) sender;

- (void) displayPingData: (NSFileHandle*) theFileHandle;
```

```
@end
```

```
MyTerminalController.m
```

```
#import "MyTerminalController.h"
```

```
@implementation MyTerminalController
```

```
- (id) init
```

```
{
    [super init];
    if (self) {
        m_pingIsRunning = FALSE;
        m_pingPipe = nil;
        m_pingTask = nil;
    }
    return self;
}
```

```
- (void) dealloc
```

```
{
    if (m_pingPipe) { [m_pingPipe release]; m_pingPipe = nil; }
    if (m_pingTask) { [m_pingTask release]; m_pingTask = nil; }
    [super dealloc];
}
```

```
// Called to initialize our fields if we need to
```

```
- (void) awakeFromNib
```

```
{
    [m_textView setString:@"Welcome to our pingy little world!"];
}
```

```
/*
```

```
ping is called when the Start Ping button is pressed.
It is also called to stop a ping in progress.
```

Essentially, this function sets things up for a separate thread that handles actually reading the data. When the user presses the Stop Ping button, the function sets a flag that triggers the thread to deallocate the task and stop the pinging.

```
*/
- (IBAction) ping: (id) sender
{
    if (m_pingIsRunning) {
        // If we are currently running and this is called,
        // we want to stop, so set a flag...
        m_pingIsRunning = FALSE;
        // ...and disable the button so it can't be clicked
        // again until we have terminated the ping
        [m_pingButton setEnabled:NO];
    } else {
        NSFileHandle *theFileHandle;

        // Otherwise we are currently not pinging so we want to start...
        // allocate a task, a pipe and the file handle
        m_pingTask = [[NSTask alloc] init];
        m_pingPipe = [[NSPipe alloc] init];
        theFileHandle = [m_pingPipe fileHandleForReading];

        if (m_pingTask && m_pingPipe && theFileHandle) {
            // If we get this far we are pretty safe so we set the global flag
            // that we are pinging...
            m_pingIsRunning = TRUE;
            // ...and begin some animation for the user to see activity
            // on the screen
            [m_pingIndicator startAnimation:self];

            // Tell the task what command (program) to execute
            [m_pingTask setLaunchPath:@"/sbin/ping"];

            // Pass some arguments to the program,
            // in this case the domain name to ping 5 times
            [m_pingTask setArguments:[NSArray arrayWithObjects:@"-c 5",
                @"triplesoft.com", nil]];

            // Set m_pingPipe as the standard output so we can see the results
            [m_pingTask setStandardOutput:m_pingPipe];

            // Launch the task
            [m_pingTask launch];

            // Clear the text we placed in the text view in awakeFromNib
            [m_textView setString:@""];

            // Create the thread that will handle reading data from
            // ping and pass theFileHandle from thePipe
            [NSThread detachNewThreadSelector:@selector(displayPingData:)
                toTarget:self withObject:theFileHandle];

            // Change the title of the button to Stop Ping to reflect
            // the change in state
        }
    }
}
```



```

// Terminate the ping task and wait for it to exit
[m_pingTask terminate];
[m_pingTask waitUntilExit];

// Check the termination status of ping, 15 or 0 means it exited successfully
// 15 = user cancelled, 0 = normal termination
if (([m_pingTask terminationStatus] != 15) &&
    ([m_pingTask terminationStatus] != 0))
    NSRunAlertPanel(@"MyTerminal", @"An error occurred trying to quit the
                        task. (%d)",
                    @"OK", @"", @"", [m_pingTask terminationStatus]);

// Release the pipe and task
if (m_pingPipe) { [m_pingPipe release]; m_pingPipe = nil; }
if (m_pingTask) { [m_pingTask release]; m_pingTask = nil; }

// Update the UI
[m_pingIndicator stopAnimation:self];
[m_pingButton setEnabled:YES];
[m_pingButton setTitle:@"Start Ping"];

// Release the autoreleasepool
[thePool release];

// Exit this thread
[NSThread exit];
}

```

/*

uptime is a simple, fast command. It only returns one line of text so we quickly are able to execute it and read the results from the pipe. No need for fancy threads or multiple reads here.

*/

```

- (IBAction)uptime: (id)sender
{

```

```

    // Allocate the task to execute and the pipe to send the output to
    NSTask          *theTask = [[NSTask alloc] init];
    NSPipe          *thePipe = [[NSPipe alloc] init];

```

```

    // Get the file handle from the pipe (assumes thePipe was allocated!)
    NSFileHandle   *theFileHandle = [thePipe fileHandleForReading];

```

```

    // Tell the task what command (program) to execute
    [theTask setLaunchPath:@"/usr/bin/uptime"];

```

```

    // Set thePipe as the standard output so we can see the results
    [theTask setStandardOutput:thePipe];

```

```

    // Launch the task
    [theTask launch];

```

```

    // Wait until the task exits, we know uptime exits immediately
    [theTask waitUntilExit];

```

```
// Verify that the program completed without error
if ([theTask terminationStatus] == 0) {
    NSString *theString;

    // Extract a string from the data returned in the pipe's file handle
    theString = [[NSString alloc]
        initWithData: [theFileHandle readDataToEndOfFile]
        encoding: NSASCIIStringEncoding];

    // Set the text field to the value of the string
    [m_textField setStringValue: theString];

    // Release what we create, theFileHandle is automatically
    // released by thePipe
    [theString release];
} else {
    // Set the text field to the value of the error
    [m_textField setIntValue: [theTask terminationStatus]];
}

[thePipe release];
[theTask release];
}

@end
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 14—MyXMLRPC

AppController.h

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

@interface AppController : NSObject
{
    IBOutlet NSPopUpButton    *m_product;
    IBOutlet NSPopUpButton    *m_color;
    IBOutlet NSTextField      *m_price;
    IBOutlet NSTextField      *m_stock;
    IBOutlet NSTextField      *m_result;
}
- (IBAction) getProductInformation: (id) sender;
@end
```

AppController.m

```
#import "AppController.h"
#import <CoreServices/CoreServices.h>

@implementation AppController

- (id) init
{
    [super init];
    if (self) {

    }
    return self;
}

- (void) dealloc
{
    [super dealloc];
}

// This is the XML RPC method that does all the work
- (IBAction) getProductInformation: (id) sender
{
    WSMethodInvocationRef    rpcCall;
    NSURL                    *rpcURL;
    NSString                 *methodName;
    NSMutableArray           *params;
    NSArray                  *paramsOrder;
    NSDictionary             *result;
    NSString                 *selectedProduct;
    NSString                 *selectedColor;

    //
```

```

//      1. Define the location of the RPC service and method
//

// Create the URL to the RPC service
rpcURL = [NSURL URLWithString:
          @"http://www.triplesoft.com/xmlrpc/product_server.php"];

// Assign the method name to call on the RPC service
methodName = @"example.getProductInfo";

// Create a method invocation
// First parameter is the URL to the RPC service
// Second parameter is the name of the RPC method to call
// Third parameter is a constant to specify the XML-RPC protocol
rpcCall = WSMethodInvocationCreate((CFURLRef)rpcURL,
                                   (CFStringRef)methodName, kWSXMLRPCProtocol);

//

//      2. Set up the parameters to be passed to the RPC method
//

// Get the users choices
selectedProduct = [m_product titleOfSelectedItem];
selectedColor = [m_color titleOfSelectedItem];

// Add the users choices to the dictionary to be passed as parameters
params = [NSMutableDictionary dictionaryWithCapacity: 2];
[params setObject:selectedProduct forKey:selectedProduct];
[params setObject:selectedColor forKey:selectedColor];

// Create the array to specify the order of the parameter values
paramsOrder = [NSArray arrayWithObjects:selectedProduct,
              selectedColor, nil];

// Set the method invocation parameters
// First parameter is the method invocation created above
// Second parameter is a dictionary containing the parameters themselves
// Third parameter is an array specifying the order of the parameters
WSMethodInvocationSetParameters(rpcCall, (CFDictionaryRef)params,
                               (CFArrayRef)paramsOrder);

//

//      3. Make the call and parse the results!
//

// Invoke the method which returns a dictionary of results
result = (NSDictionary*)WSMethodInvocationInvoke(rpcCall);

// If the results are a fault, display an error to the user with the
// fault code and descriptive string
if (WSMethodResultIsFault((CFDictionaryRef)result)) {
    NSRunAlertPanel([NSString stringWithFormat:@"Error %@",
        [result objectForKey:(NSString*)kWSFaultCode]],
        [result objectForKey:(NSString*)kWSFaultString], @"OK", @"", @"");
} else {
    // Otherwise, pull the results from the dictionary as an array

```

```
NSArray *array = [result objectForKey:
    (NSString*)kWSMethodInvocationResult];

// Display the entire array result from the server
[m_result setStringValue: [array description]];

// Display the specific fields we are interested in (price, stock)
[m_price setStringValue: [array objectAtIndex: 2]];
[m_stock setStringValue: [array objectAtIndex: 3]];
}
```

```
// Release those items that need to be released
[params release];
params = nil;
[paramsOrder release];
paramsOrder = nil;
[result release];
result = nil;
}
```

```
// Causes the application to quit when the one and only window is closed
- (BOOL) applicationShouldTerminateAfterLastWindowClosed: (NSApplication *) sender
{
    return TRUE;
}
```

@end

main.m

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

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

product_client.php

```
<html >
<head><title>PHP XML RPC Example</title></head>
<body>
<basefont size="2" face="Verdana, Arial ">

<?php
include("xmlrpc.inc");

// Show the results of the previous query if the
// parameter there are results to show
if ((($HTTP_POST_VARS["product_name"] != "") &&
    ($HTTP_POST_VARS["product_color"] != "")) {

    // Create the message object using the RPC name, parameter and its type
```

```

$message = new xmlrpcmsg("example.getProductInfo",
    array(new xmlrpcval($HTTP_POST_VARS["product_name"], "string"),
        new xmlrpcval($HTTP_POST_VARS["product_color"], "string")));

// Display the message detail
print "<pre><b>Message Detail:<br /></b>" .
    htmlentities($message->serialize()) . "</pre>";

// Create the client object which points to the server and PHP script to
// contact
$client = new xmlrpc_client("/xmlrpc/product_server.php",
    "www.triplesoft.com", 80);

// Enable debug mode so we see all there is to see coming and going
// $client->setDebug(1);
// print "<pre><b>From Server:<br /></b></pre>";

// Send the message to the server for processing
$response = $client->send($message);

// If no response was returned there was a fatal error
// (no network connection, no server, etc.)
if (!$response) { die("Send failed."); }

// If no error, display the results
if (!$response->faultCode()) {

    // Extract the values from the response
    $result_array = $response->value();
    if ($result_array->arraysize() == 4) {
        $product_name = $result_array->arraymem(0);
        $product_color = $result_array->arraymem(1);
        $product_price = $result_array->arraymem(2);
        $product_stock = $result_array->arraymem(3);

        print "<br />";
        print "<br />Product: <b>" . $product_name->scal arval () . "</b>";
        print "<br />Color: <b>" . $product_color->scal arval () . "</b>";
        print "<br />Price: <b>$" . $product_price->scal arval () . "</b>";
        print "<br />In-Stock: <b>" . $product_stock->scal arval () . "</b>";
        print "<br />";

    } else {

        print "<br />";
        print "<br />Incorrect number of items in array.
            Should be 4 but there are <b>"
            . $result_array->arraysize() . "</b>.";
        print "<br />";
    }

    // Display the response detail
    print "<pre><b>Response Detail:<br /></b>" .
        htmlentities($response->serialize()) . "</pre>";

} else {

```

```

    print "<br />";
    print "<br /><b>Fault Code:</b> " . $response->faultCode();
    print "<br /><b>Fault String:</b> " . $response->faultString();
    print "<br />";
}

```

```
print "<hr>";
```

```

}
?>

```

```

<form action="<?php echo "$PHP_SELF";?>" method="POST">
  <select name="product_name">
    <option label="Book Bag"
      value="Book Bag" selected>Book Bag</option>
    <option label="Garment Bag"
      value="Garment Bag">Garment Bag</option>
    <option label="Grocery Bag"
      value="Grocery Bag">Grocery Bag</option>
    <option label="Hand Bag"
      value="Hand Bag">Hand Bag</option>
    <option label="Old Bag"
      value="Old Bag">Old Bag</option>
  </select>
  in
  <select name="product_color">
    <option label="Black" value="Black" selected>Black</option>
    <option label="Blue" value="Blue">Blue</option>
    <option label="Brown" value="Brown">Brown</option>
    <option label="Green" value="Green">Green</option>
    <option label="Red" value="Red">Red</option>
    <option label="White" value="White">White</option>
  </select>
  <button type="submit" name="Get Product Information"
    value="Get Product Information">Get Product Information</button>
</form>

</body>
</html>

```

product_server.php

```

<?php
include("xmlrpc.inc");
include("xmlrpcs.inc");

// Array of products and prices, in real-life you might use a SQL database
$products = array (
  "Grocery Bag" => "9.00",
  "Book Bag" => "18.00",
  "Hand Bag" => "32.00",
  "Garment Bag" => "48.00",
  "Old Bag" => "0.31"
);

```

```

// Signature of this RPC
$products_signature = array(array($xml_rpcArray,
    // Returns a product price and quantity
    $xml_rpcString, // Accepts a product name
    $xml_rpcString)); // Accepts a color

// Documentation string of this RPC
$products_docstring = "When passed valid product info, more info is returned.";

// RPC Main
function get_product_info($input)
{
    global $xml_rpcerruser, $products;
    $err = "";

    // Get the parameters
    $param0 = $input->getParam(0);
    $param1 = $input->getParam(1);

    // Verify value and type
    if ((isset($param0) && ($param0->scalartyp() == "string")) &&
        (isset($param1) && ($param1->scalartyp() == "string"))) {

        // Extract parameter values
        $product_name = $param0->scalerval();
        $product_color = $param1->scalerval();

        // Attempt to find the product and price
        foreach ($products as $key => $element) {
            if ($key == $product_name) {
                $product_price = $element;
            }
        }

        // For this demo, all Browns are out of stock and all other colors
        // produce a random in-stock quantity - probably wouldn't work like
        // this in real-life, you would probably use a SQL database instead
        if ($product_color == "Brown") {
            $product_stock = 0;
        } else {
            $product_stock = rand(1, 100);
        }

        // If not found, report error
        if (!isset($product_price)) {
            $err = "No product named '" . $product_name . "' found.";
        }

    } else {
        $err = "Two string parameters (product name and color) are required.";
    }

    // If an error was generated, return it, otherwise return the proper data
    if ($err) {
        return new xml_rpcresp(0, $xml_rpcerruser, $err);
    } else {

```

```
// Create an array and fill it with the data to return
$result_array = new xmlrpcval(array(), "array");
$result_array->addScalar($product_name, "string");
$result_array->addScalar($product_color, "string");
$result_array->addScalar($product_price, "string");
$result_array->addScalar($product_stock, "string");
return new xmlrpcres($result_array);
}
}

// Create the server
$server = new xmlrpc_server(
    array("example.getProductInfo" =>
        array("function" => "get_product_info",
            "signature" => $products_signature,
            "docstring" => $products_docstring),
    )
);
?>
```

[\[Team Lib \]](#)

◀ PREVIOUS

NEXT ▶

Chapter 15—MySOAP

AppController.h

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

@interface AppController : NSObject
{
    IBOutlet NSPopUpButton *m_product;
    IBOutlet NSPopUpButton *m_color;
    IBOutlet NSTextField *m_price;
    IBOutlet NSTextField *m_stock;
    IBOutlet NSTextField *m_result;
}
- (IBAction) getProductInformation: (id) sender;
@end
```

AppController.m

```
#import "AppController.h"
#import <CoreServices/CoreServices.h>

@implementation AppController

- (id) init
{
    [super init];
    if (self) {

    }
    return self;
}

- (void) dealloc
{
    [super dealloc];
}

// This is the SOAP method that does all the work
- (IBAction) getProductInformation: (id) sender
{
    WSMethodInvocationRef    soapCall;
    NSURL                    *soapURL;
    NSString                 *methodName;
    NSMutableDictionary      *params;
    NSArray                  *paramsOrder;
    NSDictionary             *result;
    NSString                 *selectedProduct;
    NSString                 *selectedColor;

    //
```

```

//      1. Define the location of the SOAP service and method
//

// Create the URL to the SOAP service
soapURL = [NSURL URLWithString:
    @"http://www.triplesoft.com/soap/product_server.php"];

// Assign the method name to call on the SOAP service
methodName = @"get_product_info";

// Create a method invocation
// First parameter is the URL to the SOAP service
// Second parameter is the name of the SOAP method to call
// Third parameter is a constant to specify the SOAP2001 protocol
soapCall = WSMethodInvocationCreate((CFURLRef) soapURL,
    (CFStringRef) methodName, kWSSOAP2001Protocol);

//

//      2. Set up the parameters to be passed to the SOAP method
//

// Get the users choices
selectedProduct = [m_product titleOfSelectedItem];
selectedColor = [m_color titleOfSelectedItem];

// Add the users choices to the dictionary to be passed as parameters
params = [NSMutableDictionary dictionaryWithCapacity: 2];
[params setObject:selectedProduct forKey:@"product_name"];
[params setObject:selectedColor forKey:@"product_color"];

// Create the array to specify the order of the parameter values
paramsOrder = [NSArray arrayWithObjects:@"product_name",
    @"product_color", nil];

// Set the method invocation parameters
// First parameter is the method invocation created above
// Second parameter is a dictionary containing the parameters themselves
// Third parameter is an array specifying the order of the parameters,
// sometimes optional for SOAP
WSMethodInvocationSetParameters(soapCall, (CFDictionaryRef) params,
    (CFArrayRef) paramsOrder);

//

//      3. Make the call and parse the results!
//

// Invoke the method which returns a dictionary of results
result = (NSDictionary*) WSMethodInvocationInvoke(soapCall);

// If the results are a fault, display an
// error to the user with the fault code
// and descriptive string
if (WSMethodResultIsFault((CFDictionaryRef) result)) {
    NSRunAlertPanel([NSString stringWithFormat:@"Error %@",
        [result objectForKey: (NSString*) kWSFaultCode]],
        [result objectForKey: (NSString*) kWSFaultString], @"OK", @"", @"");
}

```

```

} else {
    // Otherwise, pull the results from the dictionary, held as another
    // dictionary named "soapVal"
    NSDictionary *dictionary = [result objectForKey:
        (NSString*)kWSMethodInvocationResult];
    NSDictionary *soapVal = [dictionary objectForKey:@"soapVal"];

    // Display the entire dictionary result from the server
    [m_result setStringValue: [dictionary description]];

    // Display the specific fields we are interested in (price, stock)
    [m_price setStringValue: [soapVal objectForKey:@"product_price"]];
    [m_stock setStringValue: [soapVal objectForKey:@"product_stock"]];
}

```

```

// Release those items that need to be released
[params release];
params = nil;
[paramsOrder release];
paramsOrder = nil;
[result release];
result = nil;
}

```

```

// Causes the application to quit when the one and only window is closed
- (BOOL) applicationShouldTerminateAfterLastWindowClosed: (NSApplication *) sender
{
    return TRUE;
}

```

@end

main.m

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

```

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

```

product_client.php

```

<html >
<head><title>PHP SOAP Example</title></head>
<body>
<basefont size="2" face="Verdana, Arial ">

```

```

<?php
include("nusoap.php");

```

```

// Show the results of the previous query if
// the parameter there are results to show

```

```

if (($HTTP_POST_VARS["product_name"] != "") &&
    ($HTTP_POST_VARS["product_color"] != "")) {

    // Create the parameters array
    $params = array("product_name" => $HTTP_POST_VARS["product_name"],
        "product_color" => $HTTP_POST_VARS["product_color"]);

    // Create a new soap client endpoint specifying the server location
    $client = new
        soapclient("http://www.triplesoft.com/soap/product_server.php");

    // Enable debugging
    $client->debug_flag = true;

    // Call the function, passing the parameter array
    $response = $client->call("get_product_info", $params);

    // Echo request detail
    echo "<pre><b>Request Detail:<br /></b><xmp>".
        $client->request. "</xmp></pre>";

    // If no response was returned there was a fatal error
    // (no network connection, no server, etc.)
    // You might prefer to use the $client->getError() method
    // for more detailed information.
    if (!$response) { die("Send failed."); }

    // If no error, display the results
    if (!$client->fault) {

        // Extract the values from the response
        if (count($response) == 4) {

            foreach ($response as $key => $element) {
                $$key = $element;
            }

            print "<br />";
            print "<br />Product: <b>" . $product_name . "</b>";
            print "<br />Color: <b>" . $product_color . "</b>";
            print "<br />Price: <b>$" . $product_price . "</b>";
            print "<br />In-Stock: <b>" . $product_stock . "</b>";
            print "<br />";

        } else {

            print "<br />";
            print "<br />Incorrect number of items in array.
                Should be 4 but there are <b>"
                . count($response) . "</b>.";
            print "<br />";

        }

        // Echo response detail
        echo "<pre><b>Response Detail:<br /></b><xmp>".
            $client->response. "</xmp></pre>";
    }
}

```

```

} else {
    print "<br />";
    print "<br /><b>Fault Code:</b> " . $client->faultcode;
    print "<br /><b>Fault Actor:</b> " . $client->faultactor;
    print "<br /><b>Fault String:</b> " . $client->faultstring;
    print "<br /><b>Fault Detail:</b> " . $client->faultdetail;
    // serialized?
    print "<br />";
}

// Echo debug log at the bottom since it can be large
echo "<pre><b>Debug log:<br /></b>". $client->debug_str. "</pre>";

print "<hr>";

```

```

}
?>

```

```

<form action="<?php echo "$PHP_SELF";?>" method="POST">
  <select name="product_name">
    <option label="Book Bag"
      value="Book Bag" selected>Book Bag</option>
    <option label="Garment Bag"
      value="Garment Bag">Garment Bag</option>
    <option label="Grocery Bag"
      value="Grocery Bag">Grocery Bag</option>
    <option label="Hand Bag" value="Hand Bag">Hand Bag</option>
    <option label="Old Bag" value="Old Bag">Old Bag</option>
  </select>
  in
  <select name="product_color">
    <option label="Black" value="Black" selected>Black</option>
    <option label="Blue" value="Blue">Blue</option>
    <option label="Brown" value="Brown">Brown</option>
    <option label="Green" value="Green">Green</option>
    <option label="Red" value="Red">Red</option>
    <option label="White" value="White">White</option>
  </select>
  <button type="submit" name="Get Product Information"
    value="Get Product Information">Get Product Information</button>
</form>

```

```

</body>
</html>

```

product_server.php

```

<?php
include("nusoap.php");

// Array of products and prices, in real-life you might use a SQL database
$products = array (
  "Grocery Bag" => "9.00",
  "Book Bag" => "18.00",

```

```
"Hand Bag" => "32.00",  
"Garment Bag" => "48.00",  
"Old Bag" => "0.31"  
);
```

```
// Create a new SOAP server instance  
$server = new soap_server;  
  
// Enable debugging, to be echoed via client  
$server->debug_flag = true;  
  
// Register the get_product_info function for publication  
$server->register("get_product_info");  
  
// Begin the HTTP listener service and exit  
$server->service($HTTP_RAW_POST_DATA);  
  
// Our published service  
function get_product_info($product_name, $product_color)  
{  
    global $products;  
    $err = "";  
  
    // Verify values  
    if (isset($product_name) && isset($product_color)) {  
  
        // Attempt to find the product and price  
        foreach ($products as $key => $element) {  
            if ($key == $product_name) {  
                $product_price = $element;  
            }  
        }  
  
        // For this demo, all Browns are out of stock and all other colors  
        // produce a random in-stock quantity - probably wouldn't work like  
        // this in real-life, you would probably use a SQL database instead  
        if ($product_color == "Brown") {  
            $product_stock = 0;  
        } else {  
            $product_stock = rand(1, 100);  
        }  
  
        // If not found, report error  
        if (!isset($product_price)) {  
            $err = "No product named '" . $product_name . "' found.";  
        }  
  
    } else {  
        $err = "Two string parameters (product name and color) are required.";  
    }  
  
    // If an error was generated, return it, otherwise return the proper data  
    if ($err) {  
        return new soap_fault("Client", "product_server.php", $err, $err);  
    } else {  
        // Create an array and fill it with the data to return
```

```
$result_array = array("product_name" => $product_name,  
    "product_color" => $product_color,  
    "product_price" => $product_price,  
    "product_stock" => $product_stock);  
  
return $result_array;  
}
```

```
}
```

```
exit();
```

```
?>
```

[\[Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

Appendix B. Online Resources

What follows is a list of excellent online resources to help you learn more about Mac OS X programming and related topics.

- <http://www.triplesoft.com/>— The Web site for the book you are holding in your hands. This site contains source code projects, updates, and more!
- <http://developer.apple.com/>— The Apple Developer Connection contains online documentation, sample source code, and development tool updates to help you get the most out of your Mac OS X development efforts.
- <http://www.cocoadev.com/>— This interactive Web site for Cocoa developers hosted by Steven Frank contains a lot of great information, discussion, and sample code with explanations.
- <http://cocoa.mamasam.com/>— The CocoaDev and MacOSXDev mailing list archives, which are searchable on the Web. If you are having a problem, odds are that someone has already solved it.
- <http://cocoadevcentral.com/>— Many useful tutorials for Cocoa newbies.
- <http://www.omnigroup.com/>— Developer resources, debugging utilities, and frameworks to make your development easier.
- <http://www.stepwise.com/>— Numerous articles and information for developers.
- <http://www.macosxhints.com/>— An excellent collection of (mostly) command-line–based hints for Mac OS X. Advanced users are encouraged to explore via the Terminal.
- <http://homepage2.nifty.com/hoshi-takanori/cocoa-browser/>— Cocoa Browser is a great program by Hoshi Takanori, Max Horn, and Simon Liu that allows simple and fast browsing of the Cocoa API reference documents. If you are a Cocoa programmer, you need this—it's free and is released under the GNU General Public License.

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[? conditional expression \(C programming language\) 2nd](#)

2D graphics

[rendering \(Quartz\) 2nd](#)

3D graphics

[OpenGL features](#)

[\[Team LiB \]](#)

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

About box

[scrolling](#)

[AddressBook.framework](#)

[Adobe Photoshop Web site](#) [2nd](#)

[After Dark screensaver program](#)

[agvtool utility \(command line\)](#)

alert panels

[displaying \(code snippet\)](#) [2nd](#)

[alpha channels](#) [2nd](#)

Apache

[Mac OS X preloaded software](#)

APIs

Carbon

[legacy software support](#)

APIs (Carbon)

[function of](#)

[AppController.h code \(CFPluginCocoaApp program\)](#) [2nd](#)

[AppController.h code \(MyCocoaApp program\)](#)

[AppController.h code \(MyNSBP_App program\)](#)

[AppController.h code \(MySOAP program\)](#)

[AppController.h code \(MyStatusItem program\)](#)

[AppController.h code \(MyTerminal program\)](#)

[AppController.h code \(MyThread program\)](#) [2nd](#)

[AppController.h code \(MyXMLRPC program\)](#)

[AppController.h code \(plistPlugin program\)](#)

[AppController.h code \(RadarWatcher program\)](#)

[AppController.m code \(CFPluginCocoaApp program\)](#)

[AppController.m code \(MyCocoaApp program\)](#) [2nd](#)

[AppController.m code \(MyNSBP_App program\)](#) [2nd](#)

[AppController.m code \(MySOAP program\)](#) [2nd](#) [3rd](#)

[AppController.m code \(MyStatusItem program\)](#) [2nd](#) [3rd](#) [4th](#)

[AppController.m code \(MyTerminal program\)](#) [2nd](#)

[AppController.m code \(MyThread program\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[AppController.m code \(MyXMLRPC program\)](#) [2nd](#) [3rd](#)

[AppController.m code \(plistPlugin program\)](#) [2nd](#)

[AppController.m code \(RadarWatcher program\)](#) [2nd](#)

[AppKit.framework](#)

[Apple Developer Connection Web site](#)

Apple Developer Web site

[Aqua icon design resources](#)

[localization resources](#)

[Objective-C resources](#)

Apple Developers Web site

[Objective-C resources](#)

[Apple Help Indexing Tool](#)

Apple Human Interface Guidelines

[application development](#) [2nd](#)

Apple Web site

[framework resources](#)

AppleScript

[executing via URLs](#)

[AppleScriptKit.framework](#)

Application Kit (AppKit) framework

[Cocoa development classes](#)

applications

[Aqua interface guidelines](#) [2nd](#)

Carbon

[construction example \(Project Builder\)](#)

[creating \(Project Builder\)](#)

[legacy support for pre-Mac OS X systems](#) [2nd](#)

[nib-based \(Listing 5.1\)](#) [2nd](#)

Cocoa

[calling Carbon functionality \(code snippet\)](#)

[creating](#)

[localizing](#) [2nd](#)

dock images

[updating](#) [2nd](#)

Dock menu

[updating](#) [2nd](#)

[document-based](#) [2nd](#) [3rd](#)

[RadarWatcherX project \(Cocoa\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#) [19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#) [36th](#) [37th](#)

[38th](#) [39th](#) [40th](#) [41st](#) [42nd](#) [43rd](#) [44th](#) [45th](#) [46th](#) [47th](#) [48th](#) [49th](#)

execution analysis

[Sampler utility](#)

frameworks

[Carbon, calling through](#) [2nd](#)

[Cocoa, calling through](#) [2nd](#)

icons

[creating](#) [2nd](#) [3rd](#)

installers

[creating \(PackageMaker\)](#)

Java

[creating](#)

memory allocation

[analyzing \(ObjectAlloc utility\)](#)

memory analysis

[MallocDebug utility](#)

plug-ins

[dumb](#)

[NSBundle image editing plug-in application](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#)

[plistPlugin search engine application](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

[providing capabilities](#)

RadarWatcherX project

[customizing options](#)

services

[examples](#)

[folder location](#) [2nd](#)

[function of](#)

[pasteboard reading](#) [2nd](#)

[pasteboard registration](#) [2nd](#)

[pasteboard validation](#) [2nd](#)

[pasteboard writing](#) [2nd](#)

support directories

[locating](#) [2nd](#) [3rd](#)

thread behavior

[viewing \(Thread Viewer\)](#)

threads

[function of](#) [2nd](#)

[overhead costs](#)

[parallel](#) [2nd](#)

[priorities](#) [2nd](#)

[serial](#) [2nd](#)

[versus timers](#)

user settings

[defaults](#) [command-line utility](#)

[ApplicationServices.framework](#)

Aqua interface

[appearance](#) [2nd](#) [3rd](#)

[Apple Human Interface Guidelines](#) [2nd](#)

[example](#)

AT&T

[UNIX version](#)

[AudioUnit framework](#)

[header file \(Listing 6.1\)](#) [2nd](#)

[[Team LIB](#)]

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[Berkeley Software Distribution UNIX](#), [\[See BSD UNIX\]](#)

BSD UNIX

[\(Berkeley Software Distribution UNIX\)](#)

[as basis for Mac OS X](#)

[as basis for Mac OS X system](#)

[terminal applications](#)

[bundles \(frameworks\)](#)

[\[Team LiB \]](#)

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

C

[? conditional expression](#) [2nd](#)

calling

[Carbon plug-ins](#) [2nd](#) [3rd](#)

[via Cocoa applications](#) [2nd](#) [3rd](#)

Carbon

[API software support for pre-Mac OS X systems](#)

APIs

[function of](#)

applications

[calling plug-ins](#) [2nd](#) [3rd](#)

[construction example \(Project Builder\)](#)

[creating \(Project Builder\)](#)

[calling from Cocoa applications \(code snippet\)](#)

[nib-based application example \(Listing 5.1\)](#) [2nd](#)

plug-ins

[calling via Carbon applications](#) [2nd](#) [3rd](#)

[calling via Cocoa applications](#) [2nd](#) [3rd](#)

[Carbon.framework](#)

NSBundleGetVersionNumber function

[framework version checker](#)

CFPlugin program

source code

[main.c](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

CFPlugin project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#)

[dynamic registration \(Project Builder\)](#)

[Info.plist entries \(Project Builder\)](#) [2nd](#)

[results overview](#) [2nd](#) [3rd](#)

source code

[allocMyType in main.c \(Listing 5.6\)](#)

[deallocMyType in main.c \(Listing 5.11\)](#)

[IUNKNOWN_C_GUTS in CFPluginCOM.h \(Listing 5.3\)](#)

[myAddRef in main.c \(Listing 5.9\)](#)

[myFactoryFunction in main.c \(Listing 5.5\)](#)

[myInterfaceFtbl in main.c \(Listing 5.7\)](#)

[myPlugInFunction in main.c \(Listing 5.12\)](#)

[myQueryInterface in main.c \(Listing 5.8\)](#) [2nd](#)

[myRelease in main.c \(Listing 5.10\)](#)

[MyType Definition in main.c \(Listing 5.4\)](#)

[Plug-in Interface in MyCFPlugInterface.h \(Listing 5.2\)](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#)

[static registration \(Project Builder\)](#)

[Target settings \(Project Builder\)](#)

CFPluginCarbonApp program

source code

[main.c](#) [2nd](#)

CFPluginCarbonApp project

source code

[CallPlugin in MyCFCallPlugin.c \(Listing 5.13\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

CFPluginCocoaApp program

source code

[AppController.h](#) [2nd](#)

[AppController.m](#)

[main.m](#)

CFPluginCocoaApp project

source code

[doCallPlugin in AppController.c \(Listing 5.14\)](#)

[source code \(Project Builder\)](#)

checking

[modifier key status](#) [2nd](#)

network status

[SCNetworkCheckReachabilityByName\(\) function](#) [2nd](#)

classes

CURLHandle (Cocoa)

[downloading](#)

[OmniAppKit framework \(third-party\)](#) [2nd](#)

[OmniFoundation framework \(third-party\)](#) [2nd](#)

Classic

[\(Mac OS 9\)](#) [2nd](#)

[seamless integration with Mac OS X](#) [2nd](#)

software

[running within Mac OS X](#) [2nd](#)

clients

product information project (XML-RPC)

[Cocoa implementation](#)

[Cocoa nib file](#)

[Cocoa settings](#)

[PHP implementation](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

product information system project (SOAP)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#)

Cocoa

[application development features](#)

[Java](#)

[Objective-C](#)

applications

[calling Carbon functionality \(code snippet\)](#)

[calling plug-ins](#) [2nd](#) [3rd](#)

[localizing \(NSLocalizedString macro\)](#) [2nd](#)

[based on NEXTSTEP development environment](#)

class kits

[Application Kit \(AppKit\) framework](#)

[Foundation framework](#)

CURLHandle class

[downloading](#)

[document-based applications](#) [2nd](#) [3rd](#)

[RadarWatcherX](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#)

[20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#) [36th](#) [37th](#) [38th](#) [39th](#)

[40th](#) [41st](#) [42nd](#) [43rd](#) [44th](#) [45th](#) [46th](#) [47th](#) [48th](#) [49th](#)

plug-ins

[dumb](#)

[function of](#)

[NSBundle image editing plug-in](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#)

[plistPlugin search engine application](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

product information project (XML-RPC)

[client functions](#) [2nd](#)

[nib file](#)

[project settings](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

product information system project (SOAP)

[client functions](#) [2nd](#) [3rd](#)

[nib file](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#)

[Target settings](#)

Quick functions

[using within applications](#)

[Cocoa Browser Web site](#)

[Cocoa.framework](#)

[CocoaDev.com Web site](#)

[CocoaDevCentral.com Web site](#)

Code Fragment Manager

[versioning features](#)

code listings

[MyClass.h-an Objective-C Header File \(2.1\)](#)

[MyClass.m-an Objective-C Source File \(2.2\)](#) [2nd](#)

code snippets

application's About box

[scrolling](#)

[calling Carbon from Cocoa applications](#)

[checking for modifier keys](#) [2nd](#)

[checking network status](#) [2nd](#)

[displaying alert panel](#) [2nd](#)

[displaying usernames](#)

[executing AppleScripts via URL](#)

[locating application support directories](#) [2nd](#) [3rd](#)

[locating user home directory](#)

[opening a URL](#)

QuickDraw

[drawing within Cocoa application](#)

[retrieving file icons](#)

[updating application dock image](#) [2nd](#)

[updating application Dock menu](#) [2nd](#)

color pickers

[folder location](#)

[function of](#)

MyColorPicker project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#)

[nib file \(Interface Builder\)](#) [2nd](#)

[results overview](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#)

[Target settings \(Project Builder\)](#) [2nd](#)

MyScreenSaver project

[customizing options](#) [2nd](#)

[RGB/CMYK color code](#)

Color Wheel color picker

[folder location](#)

[function of](#)

[RGB/CMYK color mode](#)

colors

[alpha channels](#) [2nd](#)

command line applications

[default Mac OS X installations](#)

[command line utilities](#) [See also [utilities](#)]

[agvtool](#)

[CpMac](#)

[DeRez](#)

[fs usage](#)

[GetFileInfo](#)

[heap](#)

[leaks](#)

[malloc history](#)

[MergePef](#)

[MvMac](#)

[ps](#)

[ResMerger](#)

[Rez](#)

[RezWack](#)

[sample](#)

[sc usage](#)

[sdp](#)

[SetFile](#)

[SplitForks](#)

[top](#)

[UnRezWack](#)

[vmmap](#)

[WSMakeStubs](#)

Concurrent Versions System (CVS)

documentation

[viewing](#)

[file control functions](#)

conditional expressions

[? \(C programming language\) 2nd](#)

copying

files

[via wildcards \(terminal applications\)](#)

[CoreFoundation.framework](#)

[CoreMIDI.framework](#)

[CoreServices.framework](#)

[CpMac utility \(command line\)](#)

[curl command-line application](#)

CURLHandle class (Cocoa)

[downloading](#)

[\[Team LIB \]](#)

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

Darwin

[FreeBSD version 4.4](#)

[Projects Directory](#)

source code

[downloading](#)

[X11 implementation](#)

debugging

applications

[gdb \(GNU Debugger\) 2nd](#)

drawings

[Quartz Debug utility](#)

defaults command-line utility

[application user settings](#)

[DeRez utility \(command line\)](#)

desktop

[windows management \(Quartz\) 2nd](#)

detecting

USB devices

[USB Prober](#)

devices

USB

[detecting \(USB Prober\)](#)

DHCP protocol

[Mac OS X support](#)

[DiscRecording.framework](#)

displaying

alert panels

[NSBeginAlertSheet\(\) function 2nd](#)

usernames

[NSUserName\(\) function](#)

DNS protocol

[Mac OS X support](#)

Dock

application images

[updating 2nd](#)

application menus

[updating](#) [2nd](#)

[preference pane](#) [2nd](#)

[document-based applications](#) [2nd](#) [3rd](#)

[RadarWatcherX \(Cocoa\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#)
[19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#) [36th](#) [37th](#) [38th](#)
[39th](#) [40th](#) [41st](#) [42nd](#) [43rd](#) [44th](#) [45th](#) [46th](#) [47th](#) [48th](#) [49th](#)

downloading

[Objective-C instruction manual](#)

drawings

[debugging \(Quartz Debug utility\)](#)

[dumb plug-ins](#)

[dynamic libraries](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

executing

[AppleScripts via URL](#)

shell scripts

[terminal applications](#)

[\[Team LiB \]](#)

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

file systems

Mac OS X support

[FTP](#)

[ISO 9660](#)

[Macintosh HFS/HFS+](#)

[MS-DOS](#)

[NFS](#)

[SMB](#)

[UFS](#)

[WebDAV](#)

FileMerge utility

[file version control functions](#)

files

[copying via wildcards \(terminal applications\)](#)

differing versions

[comparing \(FileMerge\)](#)

[find command search options](#) [2nd](#)

icons

[retrieving](#)

find command

[examples](#) [2nd](#)

[Flat File DataBase \(FFDB\) PHP Library](#)

Foundation framework

[Cocoa development classes](#)

[Foundation.framework](#)

frameworks

[AddressBook](#)

[AppKit](#)

[Apple Web site resources](#)

[AppleScriptKit](#)

[ApplicationServices](#)

[AudioUnit](#)

[header file \(Listing 6.1\)](#) [2nd](#)

[bundles](#)

[Carbon](#)

Carbon applications

[calling through](#) [2nd](#)

classes

[hiding](#)

[Cocoa](#)

Cocoa applications

[calling through](#) [2nd](#)

[CoreFoundation](#)

[CoreMIDI](#)

[CoreServices](#)

[directory structure](#) [2nd](#)

[DiscRecording](#)

[Foundation](#)

[function of](#) [2nd](#)

functions

[hiding](#)

[InterfaceBuilder](#)

[IOBluetooth](#)

[IOKit](#)

[Message](#)

MyCarbonFramework project

[constructing \(Project Builder\)](#) [2nd](#)

MyCocoaApp

[calling example](#)

[package contents](#)

MyCocoaApp project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

MyCocoaFramework project

[constructing \(Project Builder\)](#) [2nd](#)

packaging options

[application-specific](#)

[machine-specific](#)

[user-specific](#)

placement options

[application-specific](#)

[machine-specific](#)

[user-specific](#)

[PreferencePanels](#)

[ScreenSaver](#)

[similarities to libraries](#) [2nd](#)

[third-party](#)

[OmniAppKit](#) [2nd](#)

[OmniBase](#)

[OmniFoundation](#) [2nd](#)

[OmniHTML](#)

[OmniNetworking](#)

[OWF \(Omni Web Framework\)](#)

version-checking

[CFBundleGetVersionNumber](#) [function](#)

[WebCore](#)

FreeBSD (Berkeley Software Distribution)

[as basis for Darwin](#)

[fs usage utility \(command line\)](#)

FTP file system

[Mac OS X support](#)

functions

[BeginAlertSheet\(\)](#) [2nd](#)

[CFBundleGetVersionNumber](#)

[NSHomeDirectory\(\)](#)

[NSSearchPathforDirectoriesInDomain\(\)](#) [2nd](#) [3rd](#)

[NSUserName\(\)](#)

[SCNetworkCheckReachabilityByName\(\)](#) [2nd](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[gdb \(GNU Debugger\)](#)

[documentation](#)

[viewing](#) [2nd](#)

[executing](#) [2nd](#)

[GetFileInfo utility \(command line\)](#)

[Google plist.plugin](#)

[plistPlugin search engine application](#) [2nd](#)

[Google.plist code \(plistPlugin program\)](#)

[graphics](#)

[OpenGL.org Web site resources](#)

[Quartz](#)

[rendering library](#) [2nd](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[heap utility \(command line\)](#)

Help Viewer

[Indexing Tool](#)

HTTP protocol

[Mac OS X support](#)

[\[Team LiB \]](#)

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

icns Browser

[window appearance](#)

[Icon Factory Web site](#)

[IconComposer](#)

[window appearance](#)

icons

[Aqua design resources](#)

[creating](#) [2nd](#) [3rd](#)

[creating \(IconComposer utility\)](#)

creation applications

[Adobe Photoshop](#) [2nd](#)

[AquaTint](#)

[Icon Factory](#)

[IconComposer](#)

files

[retrieving](#)

[icns Browser utility](#)

[ImageWindowController.h code \(MyNSBP_App program\)](#)

[ImageWindowController.m code \(MyNSBP_App program\)](#) [2nd](#) [3rd](#) [4th](#)

installers

applications

[creating \(PackageMaker\)](#)

Interface Builder

[Cocoa application development](#)

[development functions](#)

MyColorPicker project

[nib file](#) [2nd](#)

MyNSBP_App project

[nib files](#) [2nd](#)

MyNSBP_RemoveColor project

[nib files](#) [2nd](#)

MyScreenSaver project

[nib file](#)

MyStatusItem project

[nib file](#)

MyTerminal project

[nib file](#)

RadarWatcherX project

[nib files](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[InterfaceBuilder.framework](#)

interfaces

[Aqua](#)

[appearance](#) [2nd](#) [3rd](#)

Interface Builder

[overview](#)

[pixel views \(Pixie utility\)](#)

[IOBluetooth.framework](#)

[IOKit.framework](#)

IORegistryExplorer tool

[I/O driver maintenance](#)

ISO 9660 file system

[Mac OS X support](#)

iTunes

[Open GL animation example](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[J2SE \(Java 2 Standard Edition\)](#)

Java

[Cocoa application development](#)

[Project Builder templates](#)

pure applications

[creating](#)

[Java 2 Standard Edition \(J2SE\)](#)

[Java Web Start application](#)

JavaBrowser

[window appearance](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

launching

[gdb \(GNU Debugger\)](#) [2nd](#)

LDAP protocol

[Mac OS X support](#)

[leaks utility \(command line\)](#)

libraries

[dynamic](#)

frameworks

[directory structure](#) [2nd](#)

[similarities to frameworks](#) [2nd](#)

[static](#)

Linux

[UNIX version](#)

localizing

Cocoa applications

[NSString macro](#) [2nd](#)

online resources

[Apple Developer Web site](#)

locating

[application support directories](#) [2nd](#) [3rd](#)

user home directory

[NSHomeDirectory\(\) function](#)

logons

remote

[via terminal applications](#)

[\[Team LiB \]](#)

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

Mac OS 9

[\(Classic environment\) 2nd](#)

[seamless integration with Mac OS X 2nd](#)

[Mac OS X Hints Web site](#)

Macintosh HFS/HFS+ file system

[Mac OS X support](#)

[main.c code \(CFPlugin program\) 2nd 3rd 4th 5th](#)

[main.c code \(CFPluginCarbonApp program\) 2nd](#)

[main.m code \(CFPluginCocoaApp program\)](#)

[main.m code \(MyCocoaApp program\)](#)

[main.m code \(MyNSBP_App program\)](#)

[main.m code \(MySOAP program\)](#)

[main.m code \(MyStatusItem program\)](#)

[main.m code \(MyTerminal program\)](#)

[main.m code \(MyTextService program\) 2nd](#)

[main.m code \(MyThread program\)](#)

[main.m code \(MyXMLRPC program\)](#)

[main.m code \(plistPlugin program\)](#)

[main.m code \(RadarWatcher program\)](#)

[malloc history utility \(command line\)](#)

MallocDebug utility

[memory analysis](#)

man command

[terminal applications](#)

memory

allocation analysis

[ObjectAlloc utility](#)

[analyzing \(MallocDebug utility\)](#)

[MergePef utility \(command line\)](#)

merging

files

[differing versions \(FileMerge\)](#)

[Message.framework](#)

modifier keys

status

[checking](#) [2nd](#)

MS-DOS file system

[Mac OS X support](#)

multimedia

[QuickTime features](#) [2nd](#)

[MvMac utility \(command line\)](#)

MyCarbon Framework project

source code

[MyCarbonFramework.c \(Listing 6.3\)](#)

[MyCarbonFramework.h \(Listing 6.2\)](#)

MyCarbonFramework program

source code

[MyCarbonFramework.c](#)

[MyCarbonFramework.h](#)

MyCarbonFramework project

[constructing \(Project Builder\)](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#)

[MyCarbonFramework.c code \(MyCarbonFramework program\)](#)

[MyCarbonFramework.h code \(MyCarbonFramework program\)](#)

[MyCFCallPlugin.c code \(Shared program\)](#) [2nd](#)

[MyCFPluginInterface.h code \(Shared program\)](#) [2nd](#)

MyCocoaApp

frameworks

[calling example](#)

[package contents](#)

MyCocoaApp program

source code

[AppController.h](#)

[AppController.m](#) [2nd](#)

[main.m](#)

MyCocoaApp project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

source code

[AppController Implementation in AppController.m \(Listing 6.7\)](#) [2nd](#)

[AppController Interface in AppController.h \(Listing 6.6\)](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

MyCocoaFramework program

source code

[MyCocoaFramework.h](#)

[MyCocoaFramework.m](#)

MyCocoaFramework project

[constructing \(Project Builder\)](#) [2nd](#)

source code

[MyCarbonFramework.h \(Listing 6.4\)](#)

[MyCarbonFramework.m \(Listing 6.5\)](#)

[source code \(Project Builder\)](#) [2nd](#)

[MyCocoaFramework.h code \(MyCocoaFramework program\)](#)

[MyCocoaFramework.m code \(MyCocoaFramework program\)](#)

MyColorPicker program

source code

[ThePicker.h](#)

[ThePicker.m](#) [2nd](#) [3rd](#)

MyColorPicker project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#)

[customizing options](#) [2nd](#)

[nib file \(Interface Builder\)](#) [2nd](#)

[results overview](#) [2nd](#)

source code

[ThePicker Interface in ThePicker.h \(Listing 11.1\)](#) [2nd](#)

[ThePicker](#) **D** [currentMode Method in ThePicker.m \(Listing 11.5\)](#)

[ThePicker](#) **D** [doRandomColorAction Method in ThePicker.m \(Listing 11.8\)](#) [2nd](#)

[ThePicker](#) **D** [initWithPickerMaskcolorPanel Method in ThePicker.m \(Listing 11.2\)](#) [2nd](#)

[ThePicker](#) **D** [logText Method in ThePicker.m \(Listing 11.9\)](#)

[ThePicker](#) **D** [provideNewButtonImage Method in ThePicker.m \(Listing 11.3\)](#)

[ThePicker](#) **D** [provideNewView Method in ThePicker.m \(Listing 11.7\)](#)

[ThePicker](#) **D** [setColor Method in ThePicker.m \(Listing 11.4\)](#) [2nd](#)

[ThePicker](#) **D** [supportsMode Method in ThePicker.m \(Listing 11.6\)](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#)

[Target settings \(Project Builder\)](#) [2nd](#)

[MyDocument.h code \(RadarWatcher program\)](#) [2nd](#) [3rd](#)

[MyDocument.m code \(RadarWatcher program\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

[MyNSBP Desaturate.h code \(MyNSBP_Desaturate program\)](#)

[MyNSBP Desaturate.m code \(MyNSBP_Desaturate program\)](#) [2nd](#)

[MyNSBP Protocol.h code \(MyNSBP_App program\)](#)

MyNSBP RemoveColor project

source code

[MyNSBP_Protocol Definition \(Listing 4.8\)](#)

[MyNSBP_RemoveColor +initializePluginClass/+terminatePluginClass \(Listing 4.9\)](#) [2nd](#)

[MyNSBP_RemoveColor +instantiatePlugin \(Listing 4.10\)](#)

[MyNSBP_RemoveColor -askUserWhatColorToRemove \(Listing 4.12\)](#) [2nd](#)

[MyNSBP_RemoveColor -doPlugin \(Listing 4.11\)](#) [2nd](#)

[MyNSBP_RemoveColor Interface Definition \(Listing 4.7\)](#)

[SettingsController Implementation \(Listing 4.13\)](#)

[MyNSBP RemoveColor.h code \(MyNSBP_RemoveColor program\)](#)

[MyNSBP RemoveColor.m code \(MyNSBP_RemoveColor program\)](#) [2nd](#) [3rd](#) [4th](#)

MyNSBP_App program

source code

[AppController.h](#)

[AppController.m](#) [2nd](#)

[ImageWindowController.h](#)

[ImageWindowController.m](#) [2nd](#) [3rd](#) [4th](#)

[main.m](#)

[MyNSBP Protocol.h](#)

MyNSBP_App project

[constructing \(Project Builder\)](#) [2nd](#)

[nib files \(Interface Builder\)](#) [2nd](#)

MyNSBP_Desaturate program

source code

[MyNSBP_Desaturate.h](#)

[MyNSBP_Desaturate.m](#) [2nd](#)

MyNSBP_RemoveColor program

source code

[MyNSBP_RemoveColor.h](#)

[MyNSBP_RemoveColor.m](#) [2nd](#) [3rd](#) [4th](#)

[SettingsController.h](#)

[SettingsController.m](#) [2nd](#)

MyNSBP_RemoveColor project

[building settings \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#)

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

[Info.plist entries \(Project Builder\)](#)

[nib files \(Interface Builder\)](#) [2nd](#)

[protocol adoption \(Project Builder\)](#) [2nd](#)

MyPreferencePane program

source code

[MyPreferencePanePref.h](#)

[MyPreferencePanePref.m](#) [2nd](#) [3rd](#)

MyPreferencePane project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#)

[customizing options](#)

[results overview](#) [2nd](#) [3rd](#) [4th](#)

source code

[MyPreferencePanePref Interface in MyPreferencePanePref.h \(Listing 8.1\)](#) [2nd](#)

[MyPreferencePanePref](#) **D** [buttonClicked Implementation in MyPreferencePanePref.m \(Listing 8.6\)](#)

[MyPreferencePanePref](#) **D** [checkboxClicked Implementation in MyPreferencePanePref.m \(Listing 8.5\)](#)

[MyPreferencePanePref](#) **D** [didUnselect Implementation in MyPreferencePanePref.m \(Listing 8.4\)](#) [2nd](#)

[MyPreferencePanePref](#) **D** [initWithBundle Implementation in MyPreferencePanePref.m \(Listing 8.2\)](#) [2nd](#)

[MyPreferencePanePref](#) **D** [mainViewDidLoad Implementation in MyPreferencePanePref.m \(Listing 8.3\)](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

[Target settings \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#)

[MyPreferencePanePref.h code \(MyPreferencePane program\)](#)

[MyPreferencePanePref.m code \(MyPreferencePane program\)](#) [2nd](#) [3rd](#)

MyScreenEffect program

source code

[MyScreenSaverView.h](#)

[MyScreenSaverView.m](#) [2nd](#) [3rd](#) [4th](#)

MyScreenSaver project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#)
[19th](#) [20th](#)

[customizing options](#) [2nd](#)

[nib file \(Interface Builder\)](#)

[results overview](#) [2nd](#)

source code

[MyScreenSaverView -animateOneFrame Method in MyScreenSaverView.m \(Listing 10.3\)](#) [2nd](#) [3rd](#)

[MyScreenSaverView -ConfigureSheet Method in MyScreenSaverView.m \(Listing 10.5\)](#)

[MyScreenSaverView -hasConfigureSheet Method in MyScreenSaverView.m \(Listing 10.4\)](#) [2nd](#)

[MyScreenSaverView Interface in MyScreenSaverView.h \(Listing 10.1\)](#) [2nd](#)

[MyScreenSaverView](#) **D** [closeSheetcancel Method in MyScreenSaverView.m \(Listing 10.7\)](#) [2nd](#)

[MyScreenSaverView](#) **D** [closeSheetsave Method in MyScreenSaverView.m \(Listing 10.6\)](#) [2nd](#)

[MyScreenSaverView](#) **D** [initWithFrameisPreview Method in MyScreenSaverView.m \(Listing 10.2\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[Target settings \(Project Builder\)](#)

[MyScreenSaverView.h code \(MyScreenEffect program\)](#)

[MyScreenSaverView.m code \(MyScreenEffect program\)](#) [2nd](#) [3rd](#) [4th](#)

MySOAP program

source code

[AppController.h](#)

[AppController.m](#) [2nd](#) [3rd](#)

[main.m](#)

[product client.php](#) [2nd](#) [3rd](#)

[product server.php](#) [2nd](#)

MyStatusItem program

source code

[AppController.h](#)

[AppController.m](#) [2nd](#) [3rd](#) [4th](#)

[main.m](#)

MyStatusItem project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[customizing options](#)

[nib file \(Interface Builder\)](#)

[results overview](#) [2nd](#)

source code

[AppController Interface in AppController.h \(Listing 9.1\)](#) [2nd](#)

[AppController dealloc Method in AppController.m \(Listing 9.3\)](#)

[AppController init Method in AppController.m \(Listing 9.2\)](#) [2nd](#) [3rd](#)

[AppController sellItemIP Method in AppController.m \(Listing 9.6\)](#) [2nd](#)

[AppController sellItemOne and -sellItemTwo Method in AppController.m \(Listing 9.5\)](#)

[AppController sellItemQuit Method in AppController.m \(Listing 9.7\)](#) [2nd](#)

[AppController validateMenuItem Method in AppController.m \(Listing 9.4\)](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

[Target settings \(Project Builder\)](#) [2nd](#)

MyTerminal program

source code

[AppController.h](#)

[AppController.m](#) [2nd](#)

[main.m](#)

[MyTerminalController.h](#) [2nd](#)

[MyTerminalController.m](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

MyTerminal project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#)

[nib file \(Interface Builder\)](#)

[results overview](#) [2nd](#)

source code

[MyTerminalController Interface in MyTerminalController.h \(Listing 13.1\)](#) [2nd](#)

[MyTerminalController dealloc Method in MyTerminalController.m \(Listing 13.3\)](#)

[MyTerminalController displayPingData Method in MyTerminalController.m \(Listing 13.6\)](#) [2nd](#) [3rd](#) [4th](#)

[MyTerminalController init Method in MyTerminalController.m \(Listing 13.2\)](#)

[MyTerminalController ping Method in MyTerminalController.m \(Listing 13.5\)](#) [2nd](#) [3rd](#) [4th](#)

[MyTerminalController uptime Method in MyTerminalController.m \(Listing 13.4\)](#) [2nd](#) [3rd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#)

[Target settings \(Project Builder\)](#)

[MyTerminalController.h code \(MyTerminal program\)](#) [2nd](#)

[MyTerminalController.m code \(MyTerminal program\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

MyTextService program

source code

[main.m](#) [2nd](#)

[MyTextService.h](#)

[MyTextService.m](#) [2nd](#) [3rd](#)

MyTextService project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#)

[customizing options](#) [2nd](#)

[results overview](#) [2nd](#)

source code

[MyTextService Interface in MyTextService.h \(Listing 7.2\)](#)

[MyTextService main \(Listing 7.1\)](#) [2nd](#) [3rd](#)

[MyTextService changeCase userData error in MyTextService \(Listing 7.3\)](#) [2nd](#) [3rd](#) [4th](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#)

[Target settings \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[MyTextService.h code \(MyTextService program\)](#)

[MyTextService.m code \(MyTextService program\)](#) [2nd](#) [3rd](#)

MyThread program

source code

[AppController.h](#) [2nd](#)

[AppController.m](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[main.m](#)

MyThread project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#) [19th](#) [20th](#) [21st](#) [22nd](#)

[customizing options](#) [2nd](#) [3rd](#)

[nib file \(Project Builder\)](#)

[results overview](#) [2nd](#)

source code

[AppController Interface in AppController.h \(Listing 12.1\)](#) [2nd](#)

[AppController](#) **D** [cleanupThread Method in AppController.m \(Listing 12.8\)](#) [2nd](#)

[AppController](#) **D** [start Method in AppController.m \(Listing 12.2\)](#) [2nd](#) [3rd](#)

[AppController](#) **D** [stop Method in AppController.m \(Listing 12.3\)](#)

[AppController](#) **D** [thread1 Method in AppController.m \(Listing 12.5\)](#) [2nd](#) [3rd](#) [4th](#)

[AppController](#) **D** [thread2 Method in AppController.m \(Listing 12.6\)](#) [2nd](#) [3rd](#)

[AppController](#) **D** [thread3 Method in AppController.m \(Listing 12.7\)](#) [2nd](#)

[AppController](#) **D** [windowWillClose Method in AppController.m \(Listing 12.9\)](#)

[Condition Definitions in AppController.m \(Listing 12.4\)](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#) [19th](#)

[Target settings \(Project Builder\)](#)

MyXMLRPC program

source code

[AppController.h](#)

[AppController.m](#) [2nd](#) [3rd](#)

[main.m](#)

[product client.php](#) [2nd](#) [3rd](#)

[product server.php](#) [2nd](#) [3rd](#)

[\[Team LiB \]](#)

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

native utilities

[default Mac OS X installations](#)

network protocols

[Mac OS X support](#)

Network Utility

[function of](#)

networks

[status checking](#) [2nd](#)

NEXTSTEP

[as basis for Cocoa development environment](#)

NFS file system

[Mac OS X support](#)

nib file

[MyScreenSaver project \(Interface Builder\)](#)

nib files

Carbon applications

[Listing 5.1 example](#) [2nd](#)

[NSBeginAlertSheet\(\) function](#) [2nd](#)

[NSBundle image editing plug-in application](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#)

source code

[ImageWindowController's awakeFromNib \(Listing 4.3\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[ImageWindowController's doFilter \(Listing 4.5\)](#) [2nd](#)

[ImageWindowController's doResetImage \(Listing 4.4\)](#)

[ImageWindowController's windowWillClose \(Listing 4.6\)](#) [2nd](#)

NSHomeDirectory() function

[home directory, using](#)

NSLocalizedString macro

[Cocoa application localization](#) [2nd](#)

NSMenuExtras class

[versus NSStatusItems class](#) [2nd](#) [3rd](#)

NSPreferencePane class

methods

[-assignMainView](#)

[-buttonClicked](#)

[-checkboxClicked](#)

[-didSelect](#)

[-didUnselect](#) [2nd](#)

[-firstKeyView](#)

[-initialKeyView](#)

[-initWithBundle](#)

[-lastKeyView](#)

[-mainViewDidLoad](#)

[-replyToShouldUnselect](#)

[-shouldUnselect](#)

[-willSelect](#)

[-willUnselect](#)

[NSSearchPathForDirectoriesInDomains\(\) function](#) [2nd](#) [3rd](#)

NSStatusItem class

methods

[-dealloc](#)

[-init](#) [2nd](#) [3rd](#)

[-setItemP](#)

[-setItemOne](#)

[-setItemQuit](#)

[-validateMenuItem](#)

NSStatusItems class

[versus NSMenuExtras class](#) [2nd](#) [3rd](#)

NSThread class

methods

[+setThreadPriority](#) [2nd](#)

NSUserName() function

[username display](#)

[\[Team LiB \]](#)

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

ObjectAlloc utility

[memory allocation analysis](#)

Objective-C

[Apple Developer Web site resources](#)

[classes](#)

[Cocoa application development](#)

instruction manual

[downloading](#)

[MyClass.h-an Objective-C Header File \(Listing 2.1\)](#)

[MyClass.m-an Objective-C Source File \(Listing 2.2\)](#) [2nd](#)

[symbol collisions](#) [2nd](#)

Omni Group

[third-party frameworks](#)

[OmniAppKit](#) [2nd](#)

[OmniBase](#)

[OmniFoundation](#) [2nd](#)

[OmniHTML](#)

[OmniNetworking](#)

[OWF \(Omni Web Framework\)](#)

[Web site](#)

[Omni Web Framework \(OWF\)](#)

OmniAppKit framework (third-party)

[classes](#) [2nd](#)

[OmniBase framework \(third-party\)](#)

OmniFoundation framework (third-party)

[classes](#) [2nd](#)

[OmniGroup.com Web site](#)

OmniHTML framework (third-party)

[classes](#)

[OmniNetworking framework \(third-party\)](#)

online resources

[Apple Developer Connection](#)

[Cocoa Browser](#)

[CocoaDev.com](#)

[CocoaDevCentral.com](#)

[Mac OS X Hints](#)

[OmniGroup.com](#)

[StepWise.com](#)

[TripleSoft.com](#)

OpenGL

[3D graphics library](#)

[Apple iTunes animation example](#)

[drawing features](#)

[OpenGL Info utility](#)

[OpenGL Profiler](#)

[OpenGL Shader Builder](#)

OpenGL.org Web site

[graphics resources](#)

opening

URLs

[code snippet example](#)

[Overture.plist code \(plistPlugin program\)](#)

OWF

[\(Omni Web Framework\)](#)

[\[Team LiB \]](#)

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[PackageMaker](#)

[parallel threads](#) [2nd](#)

PEFViewer

[\(Preferred Executable Format Viewer\)](#)

Perl

[Mac OS X preloaded software](#)

PHP

[Flat File DataBase \(FFDB\)](#)

[Mac OS X preloaded software](#)

product information project (XML-RPC)

[client functions](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

[server functions](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

product information system project (SOAP)

[client functions](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#)

[server functions](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

SourceForge.net Web site

[database resources](#)

[Flat File DataBase \(FFDB\) PHP Library](#)

Pixie utility

[pixel viewing functions](#)

plistPlugin program

source code

[AppController.h](#)

[AppController.m](#) [2nd](#)

[Google.plist](#)

[main.m](#)

[Overture.plist](#)

[SearchWindowController.h](#)

[SearchWindowController.m](#) [2nd](#) [3rd](#)

[Yahoo.plist](#)

[plistPlugin search engine application](#)

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#)

[Google.plist plug-in](#) [2nd](#)

[package contents](#)

source code

[-awakeFromNib \(Listing 4.1\)](#) [2nd](#)

[-doSearch \(Listing 4.2\)](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#)

[Target settings \(Project Builder\)](#) [2nd](#)

[plists \(property lists\)](#)

[plug-ins](#)

[applications](#)

[providing capabilities](#)

[Carbon](#)

[calling via applications](#) [2nd](#) [3rd](#)

[calling via Cocoa applications](#) [2nd](#) [3rd](#)

[CFPlugin project](#)

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#)

[dynamic registration \(Project Builder\)](#)

[Info.plist entries \(Project Builder\)](#) [2nd](#)

[results overview](#) [2nd](#) [3rd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#)

[static registration \(Project Builder\)](#)

[Target settings \(Project Builder\)](#)

[CFPluginCarbonApp project](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[CFPluginCocoaApp project](#)

[source code \(Project Builder\)](#)

[dumb](#)

[function of](#)

[MyNSBP_App project](#)

[constructing \(Project Builder\)](#) [2nd](#)

[nib files \(Interface Builder\)](#) [2nd](#)

[MyNSBP_RemoveColor project](#)

[build settings \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#)

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

[Info.plist entries \(Project Builder\)](#)

[nib files \(Interface Builder\)](#) [2nd](#)

[protocol adoption \(Project Builder\)](#) [2nd](#)

[NSBundle image editing application](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#)

[plistPlugin search engine application](#)

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#)

[Google.plist plug-in](#) [2nd](#)

[package contents](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#)

[Target settings \(Project Builder\)](#) [2nd](#)

[third-party development](#)

PPP protocol

[Mac OS X support](#)

preference panes

[Dock](#) [2nd](#)

[examples](#)

[folder location](#)

[function of](#) [2nd](#)

MyPreferencePane project

[customizing options](#)

[results overview](#) [2nd](#) [3rd](#)

[PreferenceController.h code \(RadarWatcher program\)](#)

[PreferenceController.m code \(RadarWatcher program\)](#) [2nd](#)

[PreferencePanes.framework](#)

[Process Viewer \(UNIX\)](#)

processes

threads

[function of](#) [2nd](#)

[overhead costs](#)

[parallel](#) [2nd](#)

[priorities](#) [2nd](#)

[serial](#) [2nd](#)

[versis timers](#)

[viewing execution of \(sample command-line tool\)](#)

[product client.php code \(MySOAP program\)](#) [2nd](#) [3rd](#)

[product client.php code \(MyXMLRPC program\)](#) [2nd](#) [3rd](#)

product information project (SOAP)

[Cocoa client](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[PHP client](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#)

[PHP server](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

product information project (XML-RPC)

[Cocoa client](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#)

[PHP client](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

[PHP server](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

[results overview](#) [2nd](#) [3rd](#)

source code

[-getProductInformation in ApplicationController.h \(Listing 14.12\)](#) [2nd](#) [3rd](#)

[-getProductInformation](#) **D** [Part 1 in ApplicationController.m \(Listing 14.13\)](#) [2nd](#)

[-getProductInformation](#) **D** [Part 2 in ApplicationController.m \(Listing 14.14\)](#) [2nd](#)

[-getProductInformation](#) **D** [Part 3 in ApplicationController.m \(Listing 14.14\)](#) [2nd](#)

[AppController Interface in ApplicationController.h \(Listing 14.11\)](#)

[Creating the Server in product server.php \(Listing 14.6\)](#)

[In the Beginning of product client.php \(Listing 14.8\)](#)

[In the End of product client.php \(Listing 14.10\)](#) [2nd](#)

[In the Middle of product client.php \(Listing 14.9\)](#) [2nd](#) [3rd](#) [4th](#)

[Initial Steps in product server.php \(Listing 14.2\)](#) [2nd](#)

[product client.php \(Listing 14.7\)](#) [2nd](#) [3rd](#)

[product server.php \(Listing 14.1\)](#) [2nd](#) [3rd](#)

[The get product info Method in product server.php \(Listing 14.3\)](#) [2nd](#)

[The XML-Formatted Data Sent from product server.php \(Listing 14.5\)](#)

[The XML-Formatted Data Sent to product server.php \(Listing 14.4\)](#) [2nd](#)

product information system project (SOAP)

[Cocoa client](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#)

[results overview](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#)
[21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#)

source code

[-getProductInformation in ApplicationController.m \(Listing 15.12\)](#) [2nd](#) [3rd](#)

[-getProductInformation](#) **D** [Part 1 in ApplicationController.m \(Listing 15.13\)](#) [2nd](#)

[-getProductInformation](#) **D** [Part 2 in ApplicationController.m \(Listing 15.14\)](#) [2nd](#)

[-getProductInformation](#) **D** [Part 3 in ApplicationController.m \(Listing 15.15\)](#) [2nd](#) [3rd](#)

[AppController Interface in ApplicationController.h \(Listing 15.11\)](#)

[In the Beginning of product client.php \(Listing 15.8\)](#)

[In the End of product client.php \(Listing 15.10\)](#) [2nd](#)

[In the Middle of product client.php \(Listing 15.9\)](#) [2nd](#) [3rd](#) [4th](#)

[Initial Steps in product server.php \(Listing 15.2\)](#) [2nd](#)

[product client.php \(Listing 15.7\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[product server.php \(Listing 15.1\)](#) [2nd](#) [3rd](#)

[The get product info Method in product server.php \(Listing 15.3\)](#) [2nd](#)

[The XML-Formatted Data Returned from product server.php \(Listing 15.5\)](#) [2nd](#)

[The XML-Formatted Data Sent to product server.php \(Listing 15.4\)](#) [2nd](#)

[product server.php code \(MySOAP program\)](#) [2nd](#)

[product server.php code \(MyXMLRPC program\)](#) [2nd](#) [3rd](#)

programming languages

Java

[Project Builder templates](#)

Objective-C

[classes](#)

[MyClass.h-an Objective-C Header File \(Listing 2.1\)](#)

[MyClass.m-an Objective-C Source File \(Listing 2.2\)](#) [2nd](#)

Project Builder

Carbon application example

[constructing](#)

CFPlugin project

[constructing](#) [2nd](#) [3rd](#) [4th](#)

[dynamic registration](#)

[Info.plist entries](#) [2nd](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#)

[static registration](#)

[Target settings](#)

CFPluginCarbonApp project

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

CFPluginCocoaApp project

[source code](#)

[Cocoa application development](#)

[Concurrent Versions System \(CVSD\)](#)

debugging utilities

[gdb](#) [2nd](#) [3rd](#) [4th](#)

[development functions](#)

frameworks

[AddressBook](#)

[AppKit](#)

[AppleScriptKit](#)

[ApplicationServices](#)

[Carbon](#)

[Cocoa](#)

[CoreFoundation](#)

[CoreMIDI](#)

[CoreServices](#)

[DiscRecording](#)

[Foundation](#)

[InterfaceBuilder](#)

[IOBluetooth](#)

[IOKit](#)

[Message](#)

[PreferencePanels](#)

[ScreenSaver](#)

[WebCore](#)

MyCarbonFramework project

[constructing](#) [2nd](#)

[source code](#) [2nd](#) [3rd](#)

MyCocoaApp project

[constructing](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

MyCocoaFramework project

[constructing](#) [2nd](#)

[source code](#) [2nd](#)

MyColorPicker project

[constructing](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#)

[Target settings](#) [2nd](#)

MyNSBP_App project

[constructing](#) [2nd](#)

MyNSBP_RemoveColor project

[building settings](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#)

[constructing](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

[Info.plist entries](#)

[protocol adoption](#) [2nd](#)

MyPreferencePane project

[constructing](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

[Target settings](#) [2nd](#) [3rd](#) [4th](#)

MyScreenSaver project

[constructing](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[Target settings](#)

MyStatusItem project

[constructing](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

[Target settings](#) [2nd](#)

MyTerminal project

[constructing](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#)

[Target settings](#)

MyTextService project

[constructing](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#)

[Target settings](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

MyThread

[constructing](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#)

[21st](#) [22nd](#)

[nib file](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#)

[Target settings](#)

plistPlugin search engine application

[constructing](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#)

RadarWatcherX project

[constructing](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#)

[21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#) [36th](#) [37th](#) [38th](#) [39th](#) [40th](#)

[41st](#) [42nd](#) [43rd](#) [44th](#)

[copyrights/credits](#) [2nd](#)

[icons](#) [2nd](#) [3rd](#)

[Info.plist entries](#) [2nd](#)

[source code](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#) [20th](#)

[21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#)

projects

CFPlugin

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#)

[dynamic registration \(Project Builder\)](#)

[Info.plist entries \(Project Builder\)](#) [2nd](#)

[results overview](#) [2nd](#) [3rd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#)

[static registration \(Project Builder\)](#)

[Target settings \(Project Builder\)](#)

CFPluginCarbonApp

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

CFPluginCocoaApp

[source code \(Project Builder\)](#)

MyCarbonFramework

[constructing \(Project Builder\)](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#)

MyCocoaApp

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

MyCocoaFramework

[constructing \(Project Builder\)](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#)

MyColorPicker

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#)

[nib file \(Interface Builder\)](#) [2nd](#)

[results overview](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#)

[Target settings \(Project Builder\)](#) [2nd](#)

MyNSBP_App

[constructing \(Project Builder\)](#) [2nd](#)

[nib files \(Interface Builder\)](#) [2nd](#)

MyNSBP_RemoveColor

[build settings \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#)

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

[Info.plist entries \(Project Builder\)](#)

[nib files \(Interface Builder\)](#) [2nd](#)

[protocol adoption \(Project Builder\)](#) [2nd](#)

MyPreferencePane

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#)

[results overview](#) [2nd](#) [3rd](#) [4th](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

[Target settings \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#)

MyScreenSaver

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#) [19th](#) [20th](#)

[nib file \(Interface Builder\)](#)

[results overview](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[Target settings \(Project Builder\)](#)

MyStatusItem

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[nib file \(Interface Builder\)](#)

[results overview](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

[Target settings \(Project Builder\)](#) [2nd](#)

MyTerminal

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#)

[nib file \(Interface Builder\)](#)

[results overview](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#)

[Target settings \(Project Builder\)](#)

MyTextService

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#)

[results overview](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#)

[Target settings \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

MyThread

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#) [19th](#) [20th](#) [21st](#) [22nd](#)

[nib file \(Project Builder\)](#)

[results overview](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#) [19th](#)

[Target settings \(Project Builder\)](#)

RadarWatcherX (Cocoa)

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#) [19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#) [36th](#) [37th](#)

[38th](#) [39th](#) [40th](#) [41st](#) [42nd](#) [43rd](#) [44th](#)

[copyrights/credits \(Project Builder\)](#) [2nd](#)

[icons \(Project Builder\)](#) [2nd](#) [3rd](#)

[Info.plist entries \(Project Builder\)](#) [2nd](#)

[nib files \(Interface Builder\)](#) [2nd](#) [3rd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#) [19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#)

SOAP product information system project

[Cocoa client](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#)

[PHP client](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#)

[PHP server](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

[results overview](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#)
[20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#)

XML-RPC product information project

[Cocoa client](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#)

[PHP client](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

[PHP server](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

[results overview](#) [2nd](#) [3rd](#)

[Projects Directory \(Darwin\)](#)

Property List Editor

XML text files

[viewing](#)

[ps utility \(command line\)](#)

[Pyro screensaver program](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

Quartz

[desktop management](#) [2nd](#)

[drawing features](#) [2nd](#)

[graphics rendering library](#) [2nd](#)

[window management](#) [2nd](#)

[Quartz Debug utility](#)

QuickDraw

[drawing within Cocoa applications](#)

QuickTime

[multimedia features](#) [2nd](#)

[multiple platform support](#) [2nd](#)

[\[Team LiB \]](#)

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[RadarView.h code \(RadarWatcher program\)](#) [2nd](#)

[RadarView.m code \(RadarWatcher program\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

RadarWatcher program

source code

[AppController.h](#)

[AppController.m](#) [2nd](#)

[main.m](#)

[MyDocument.h](#) [2nd](#) [3rd](#)

[MyDocument.m](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

[PreferenceController.h](#)

[PreferenceController.m](#) [2nd](#)

[RadarView.h](#) [2nd](#)

[RadarView.m](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

RadarWatcherX project

[customizing options](#)

document data

[reading/writing \(NSMutableDictionary\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#)

[storing \(NSMutableDictionary\)](#) [2nd](#)

source code

[AppController +initialize \(Listing 3.15\)](#) [2nd](#)

[AppController](#) [D](#) [applicationShoudlOpenUntitledFile \(Listing 3.16\)](#) [2nd](#)

[AppController](#) [D](#) [showPreferencePanel \(Listing 3.17\)](#)

[application preferences](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)

[Creating a Timer \(Listing 3.8\)](#)

[Main.m \(Listing 3.1\)](#) [2nd](#) [3rd](#)

[MyDocument](#) [D](#) [RadarwatcherX .rdrw Data File \(Listing 3.7\)](#)

[MyDocument](#) [D](#) [init \(Listing 3.3\)](#)

[MyDocument](#) [D](#) [RadarwatcherX .rdrw Data File \(Listing 3.7\)](#)

[MyDocument](#) [D](#) [refreshRadar method Image Checking \(Listing 3.11\)](#) [2nd](#)

[MyDocument](#) [D](#) [refreshRadar method Image Loaded \(Listing 3.10\)](#) [2nd](#)

[MyDocument](#) [D](#) [refreshRadar method Image Loading \(Listing 3.9\)](#)

[MyDocument](#) [D](#) [updateUI \(Listing 3.6\)](#) [2nd](#)

[MyDocument](#) [D](#) [writeFromFileofType \(Listing 3.5\)](#)

[MyDocument](#) [D](#) [writeToFileofType \(Listing 3.4\)](#) [2nd](#) [3rd](#)

[NSMutableDictionary Keys \(Listing 3.2\)](#)

[PreferenceController.h \(Listing 3.14\)](#)

[PreferenceController.m \(Listing 3.18\)](#) [2nd](#)

[RadarView -isColorInImageInRect \(Listing 3.13\)](#) [2nd](#) [3rd](#)

[RadarView color checker](#) [2nd](#) [3rd](#)

[RadarView instance variable](#) [2nd](#)

[RadarView Mouse Overrides \(Listing 3.12\)](#) [2nd](#)

[RadarView watch box](#) [2nd](#)

RadarWatcherX project (Cocoa)

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#)
[19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#) [36th](#) [37th](#) [38th](#)
[39th](#) [40th](#) [41st](#) [42nd](#) [43rd](#) [44th](#)

[copyrights/credits \(Project Builder\)](#) [2nd](#)

[document window](#) [2nd](#)

[icons \(Project Builder\)](#) [2nd](#) [3rd](#)

[Info.plist entries \(Project Builder\)](#) [2nd](#)

[nib files \(Interface Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#)
[19th](#) [20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#) [30th](#) [31st](#) [32nd](#) [33rd](#) [34th](#) [35th](#)

reading

[pasteboard types](#)

[services](#) [2nd](#)

registering

[pasteboard types](#)

[services](#) [2nd](#)

remote logons

[terminal applications](#)

[ResMerger utility \(command line\)](#)

resources

online

[Apple Developer Connection Web site](#)

[Cocoa Browser Web site](#)

[CocoaDev.com](#)

[CocoaDevCentral.com](#)

[Mac OS X Hints Web site](#)

[OmniGroup.com](#)

[StepWise.com](#)

[TripleSoft.com](#)

[Rez utility \(command line\)](#)

[RezWack utility \(command line\)](#)

rm command

[terminal applications](#)

[\[Team LiB \]](#)

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

sample command-line tool

[report elements](#)

[viewing process execution](#)

[sample utility \(command line\)](#)

Sampler utility

[application execution analysis](#)

[sc usage utility \(command line\)](#)

[SCNetworkCheckReachabilityByName\(\) function](#) [2nd](#)

screen effects

MyScreenSaver project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#) [19th](#) [20th](#)

[customizing options](#) [2nd](#)

[nib file \(Interface Builder\)](#)

[results overview](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[Target settings \(Project Builder\)](#)

[OpenGL environment resources](#)

[screensaver terminology](#)

[selection options](#) [2nd](#) [3rd](#)

Screen Effects pane

[\(System Preferences\)](#)

[Activation tab](#)

[Hot Corners tab](#)

[Screen Effects tab](#)

[ScreenSaver.framework](#)

[screensavers](#) [See also [screen effects](#)]

[After Dark](#)

[Pyro](#)

scrolling

[application About box](#)

[sdp utility \(command line\)](#)

searching

files

[find command options](#) [2nd](#)

[SearchWindowController.h code \(plistPlugin program\)](#)

[SearchWindowController.m code \(plistPlugin program\)](#) [2nd](#) [3rd](#)

sendmail

[Mac OS X preloaded software](#)

[serial threads](#) [2nd](#)

servers

product information project (XML-RPC)

[PHP implementation](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

UNIX

[stability of](#)

services

[\(System Services\)](#)

[examples](#)

[folder location](#) [2nd](#)

[function of](#)

MyPreferencePane project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#)

[results overview](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

[Target settings \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#)

MyTextService project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#)

[customizing options](#) [2nd](#)

[results overview](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#)

[Target settings \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

pasteboard types

[reading](#) [2nd](#)

[registering](#) [2nd](#)

[validating](#) [2nd](#)

[writing](#) [2nd](#)

source code

[Reading Pasteboard Types \(Listing 7.7\)](#) [2nd](#)

[Registering Pasteboard Types \(Listing 7.4\)](#) [2nd](#)

[Validating Pasteboard Types \(Listing 7.5\)](#) [2nd](#)

[Writing Pasteboard Types \(Listing 7.6\)](#) [2nd](#)

[SetFile utility \(command line\)](#)

[SettingsController.h code \(MyNSBP_RemoveColor program\)](#)

[SettingsController.m code \(MyNSBP_RemoveColor program\)](#) [2nd](#)

Shared program

source code

[MyCFCallPlugin.c](#) [2nd](#)

[MyCFPluginInterface.h](#) [2nd](#)

shell scripts

[executing \(terminal applications\)](#)

[Simple Object Access Protocol, _](#) [See [SOAP](#)]

SMB file system

[Mac OS X support](#)

snippets (code)

[calling Carbon from Cocoa applications](#)

[checking for modifier keys](#) [2nd](#)

[checking network status](#) [2nd](#)

[displaying alert panel](#) [2nd](#)

[displaying usernames](#)

[executing AppleScripts via URL](#)

file icons

[retrieving](#)

[locating application support directories](#) [2nd](#) [3rd](#)

[locating user home directory](#)

[opening a URL](#)

QuickDraw

[drawing within Cocoa application](#)

[scrolling application's About box](#)

[updating application dock image](#) [2nd](#)

[updating application Dock menu](#) [2nd](#)

SOAP

[\(Simple Object Access Protocol\)](#)

[features](#) [2nd](#) [3rd](#)

product information system project

[Cocoa client](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#)

[PHP client](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#)

[PHP server](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

[results overview](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#) [18th](#) [19th](#)

[20th](#) [21st](#) [22nd](#) [23rd](#) [24th](#) [25th](#) [26th](#) [27th](#) [28th](#) [29th](#)

[W3C Web site specification resources](#)

SOAP protocol

[Mac OS X support](#)

software

Carbon

[legacy support for pre-Mac OS X systems](#)

Classic (Mac OS 9)

[running under Mac OS X system](#) [2nd](#)

[default Mac OS X installations](#)

[preinstalled](#)

source code

[AppController Implementation in AppController.m \(Listing 6.7\)](#) [2nd](#)

[AppController Interface in AppController.h \(Listing 12.1\)](#) [2nd](#)

[AppController Interface in AppController.h \(Listing 6.6\)](#) [2nd](#)

[AppController Interface in AppController.h \(Listing 9.1\)](#) [2nd](#)

[AppController](#) **D** [cleanupThread Method in AppController.m \(Listing 12.8\)](#) [2nd](#)

[AppController](#) **D** [dealloc Method in AppController.m \(Listing 9.3\)](#)

[AppController](#) **D** [init Method in AppController.m \(Listing 9.2\)](#) [2nd](#) [3rd](#)

[AppController](#) **D** [setItemIP Method in AppController.m \(Listing 9.6\)](#) [2nd](#)

[AppController](#) **D** [setItemOne and](#) **D** [setItemTwo Method in AppController.m \(Listing 9.5\)](#)

[AppController](#) **D** [setItemQuit Method in AppController.m \(Listing 9.7\)](#) [2nd](#)

[AppController](#) **D** [start Method in AppController.m \(Listing 12.2\)](#) [2nd](#) [3rd](#)

[AppController](#) **D** [stop Method in AppController.m \(Listing 12.3\)](#)

[AppController](#) **D** [thread1 Method in AppController.m \(Listing 12.5\)](#) [2nd](#) [3rd](#) [4th](#)

[AppController](#) **D** [thread2 Method in AppController.m \(Listing 12.6\)](#) [2nd](#) [3rd](#)

[AppController](#) **D** [thread3 Method in AppController.m \(Listing 12.7\)](#) [2nd](#)

[AppController](#) **D** [validateMenuItem Method in AppController.m \(Listing 9.4\)](#)

[AppController](#) **D** [windowWillClose Method in AppController.m \(Listing 12.9\)](#)

[AudioUnit framework header file \(Listing 6.1\)](#) [2nd](#)

Carbon applications

[nib-based \(Listing 5.1\)](#) [2nd](#)

CFPlugin program

[main.c](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

CFPlugin project

[allocMyType in main.c \(Listing 5.6\)](#)

[deallocMyType in main.c \(Listing 5.11\)](#)

[IUNKNOWN_C_GUTS in CFPluginCOM.h \(Listing 5.3\)](#)

[myAddRef in main.c \(Listing 5.9\)](#)

[myFactoryFunction in main.c \(Listing 5.5\)](#)

[myInterfaceFtbl in main.c \(Listing 5.7\)](#)

[myPlugInFunction in main.c \(Listing 5.12\)](#)

[myQueryInterface in main.c \(Listing 5.8\)](#) [2nd](#)

[myRelease in main.c \(Listing 5.10\)](#)

[MyType Definition in main.c \(Listing 5.4\)](#)

[Plug-in Interface in MyCFPluginInterface.h \(Listing 5.2\)](#) [2nd](#)

CFPluginCarbonApp program

[main.c](#) [2nd](#)

CFPluginCarbonApp project

[CallPlugin in MyCFCallPlugin.c \(Listing 5.13\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

CFPluginCocoaApp program

[AppController.h](#) [2nd](#)

[AppController.m](#)

[main.m](#)

CFPluginCocoaApp project

[doCallPlugin in AppController.c \(Listing 5.14\)](#)

[Condition Definitions in AppController.m \(Listing 12.4\)](#)

Darwin

[downloading](#)

MyCarbonFramework program

[MyCarbonFramework.c](#)

[MyCarbonFramework.h](#)

[MyCarbonFramework.c \(Listing 6.3\)](#)

[MyCarbonFramework.h \(Listing 6.2\)](#)

MyCocoaApp program

[AppController.h](#)

[AppController.m](#) [2nd](#)

[main.m](#)

MyCocoaFramework program

[MyCocoaFramework.h](#)

[MyCocoaFramework.m](#)

[MyCocoaFramework.h \(Listing 6.4\)](#)

[MyCocoaFramework.m \(Listing 6.5\)](#)

MyColorPicker program

[ThePicker.h](#)

[ThePicker.m](#) [2nd](#) [3rd](#)

MyNSBP RemoveColor project

[MyNSBP_Protocol Definition \(Listing 4.8\)](#)

[MyNSBP_RemoveColor +initializePluginClass/+terminatePluginClass \(Listing 4.9\)](#) [2nd](#)

[MyNSBP_RemoveColor +instantiatePlugin \(Listing 4.10\)](#)

[MyNSBP_RemoveColor -askUserWhatColorToRemove \(Listing 4.12\)](#) [2nd](#)

[MyNSBP_RemoveColor -doPlugin \(Listing 4.11\)](#) [2nd](#)

[MyNSBP_RemoveColor Interface Definition \(Listing 4.7\)](#)

[SettingsController Implementation \(Listing 4.13\)](#)

MyNSBP_App program

[AppController.h](#)

[AppController.m](#) [2nd](#)

[ImageWindowController.h](#)

[ImageWindowController.m](#) [2nd](#) [3rd](#) [4th](#)

[main.m](#)

[MyNSBP Protocol.h](#)

MyNSBP_Desaturate program

[MyNSBP_Desaturate.h](#)

[MyNSBP_Desaturate.m](#) [2nd](#)

MyNSBP_RemoveColor program

[MyNSBP_RemoveColor.h](#)

[MyNSBP_RemoveColor.m](#) [2nd](#) [3rd](#) [4th](#)

[SettingsController.h](#)

[SettingsController.m](#) [2nd](#)

MyPreferencePane program

[MyPreferencePanePref.h](#)

[MyPreferencePanePref.m](#) [2nd](#) [3rd](#)

[MyPreferencePanePref Interface in MyPreferencePanePref.h \(Listing 8.1\)](#) [2nd](#)

[MyPreferencePanePref](#) **D** [buttonClicked Implementation in MyPreferencePanePref.m \(Listing 8.6\)](#)

[MyPreferencePanePref](#) **D** [checkboxClicked Implementation in MyPreferencePanePref.m \(Listing 8.5\)](#)

[MyPreferencePanePref](#) **D** [didUnselect Implementation in MyPreferencePanePref.m \(Listing 8.4\)](#) [2nd](#)

[MyPreferencePanePref](#) **D** [initWithBundle Implementation in MyPreferencePanePref.m \(Listing 8.2\)](#) [2nd](#)

[MyPreferencePanePref](#) **D** [mainViewDidLoad Implementation in MyPreferencePanePref.m \(Listing 8.3\)](#) [2nd](#)

MyScreenEffect program

[MyScreenSaverView.h](#)

[MyScreenSaverView.m](#) [2nd](#) [3rd](#) [4th](#)

[MyScreenSaverView Interface in MyScreenSaverView.h \(Listing 10.1\)](#) [2nd](#)

[MyScreenSaverView](#) **D** [animateOneFrame Method in MyScreenSaverView.m \(Listing 10.3\)](#) [2nd](#) [3rd](#)

[MyScreenSaverView](#) **D** [closeSheetcancel Method MyScreenSaverView.m \(Listing 10.7\)](#) [2nd](#)

[MyScreenSaverView](#) **D** [closeSheetsave Method MyScreenSaverView.m \(Listing 10.6\)](#) [2nd](#)

[MyScreenSaverView](#) **D** [ConfigureSheet Method in MyScreenSaverView.m \(Listing 10.5\)](#)

[MyScreenSaverView](#) **D** [hasConfigureSheet Method in MyScreenSaverView.m \(Listing 10.4\)](#) [2nd](#)

[MyScreenSaverView](#) **D** [initWithFrameisPreview Method in MyScreenSaverView.m \(Listing 10.2\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

MySOAP program

[AppController.h](#)

[AppController.m](#) [2nd](#) [3rd](#)

[main.m](#)

[product client.php](#) [2nd](#) [3rd](#)

[product server.php](#) [2nd](#)

MyStatusItem program

[AppController.h](#)

[AppController.m](#) [2nd](#) [3rd](#) [4th](#)

[main.m](#)

MyTerminal program

[AppController.h](#)

[AppController.m](#) [2nd](#)

[main.m](#)

[MyTerminalController.h](#) [2nd](#)

[MyTerminalController.m](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)

[MyTerminalController](#) Interface in [MyTerminalController.h \(Listing 13.1\)](#) [2nd](#)

[MyTerminalController](#) **D** [dealloc Method in MyTerminalController.m \(Listing 13.2\)](#)

[MyTerminalController](#) **D** [displayPingData Method in MyTerminalController.m \(Listing 13.6\)](#) [2nd](#) [3rd](#) [4th](#)

[MyTerminalController](#) **D** [init Method in MyTerminalController.m \(Listing 13.2\)](#)

[MyTerminalController](#) **D** [ping Method in MyTerminalController.m \(Listing 13.5\)](#) [2nd](#) [3rd](#) [4th](#)

[MyTerminalController](#) **D** [uptime Method in MyTerminalController.m \(Listing 13.4\)](#) [2nd](#) [3rd](#)

[MyTextService](#) Interface in [MyTextService.h \(Listing 7.2\)](#)

[MyTextService](#) [main \(Listing 7.1\)](#) [2nd](#) [3rd](#)

MyTextService program

[main.m](#) [2nd](#)

[MyTextService.h](#)

[MyTextService.m](#) [2nd](#) [3rd](#)

[MyTextService](#) **D** [changeCase userData error in MyTextService.m \(Listing 7.3\)](#) [2nd](#) [3rd](#) [4th](#)

MyThread program

[AppController.h](#) [2nd](#)

[AppController.m](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[main.m](#)

MyXMLRPC program

[AppController.h](#)

[AppController.m](#) [2nd](#) [3rd](#)

[main.m](#)

[product client.php](#) [2nd](#) [3rd](#)

[product server.php](#) [2nd](#) [3rd](#)

NSBundle image editing plug-in applications

[ImageWindowsController's -windowWillClose \(Listing 4.6\)](#) [2nd](#)

[ImageWindowsController's awakeFromNib \(Listing 4.3\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

[ImageWindowsController's doFilter \(Listing 4.5\)](#) [2nd](#)

[ImageWindowsController's doResetImage \(Listing 4.4\)](#)

plistPlugin application

[-awakeFromNib \(Listing 4.1\)](#) [2nd](#)

[-doSearch \(Listing 4.2\)](#) [2nd](#)

plistPlugin program

[AppController.h](#)

[AppController.m](#) [2nd](#)

[Google.plist](#)

[main.m](#)

[Overture.plist](#)

[SearchWindowController.h](#)

[SearchWindowController.m](#) [2nd](#) [3rd](#)

[Yahoo.plist](#)

product information product (XML-RPC)

[Creating the Server in product server.php \(Listing 14.6\)](#)

[Initial Steps in product server.php \(Listing 14.2\)](#) [2nd](#)

[product server.php \(Listing 14.1\)](#) [2nd](#) [3rd](#)

[The get product info Method in product server.php \(Listing 14.3\)](#) [2nd](#)

[The XML-Formatted Data Sent from product server.php \(Listing 14.5\)](#)

[The XML-Formatted Data Sent to product server.php \(Listing 14.4\)](#) [2nd](#)

product information product (XML-RPC) project

[In the Beginning of product client.php \(Listing 14.8\)](#)

[In the End of product client.php \(Listing 14.10\)](#) [2nd](#)

[In the Middle of product client.php \(Listing 14.9\)](#) [2nd](#) [3rd](#) [4th](#)

[product client.php \(Listing 14.7\)](#) [2nd](#) [3rd](#)

product information project (XML-RPC)

[-getProductInformation in AppController.h \(Listing 14.12\)](#) [2nd](#) [3rd](#)

[-getProductInformation](#) **D** [Part 1 in ApplicationController.m \(Listing 14.13\)](#) [2nd](#)
[-getProductInformation](#) **D** [Part 2 in ApplicationController.m \(Listing 14.14\)](#) [2nd](#)
[-getProductInformation](#) **D** [Part 3 in ApplicationController.m \(Listing 14.15\)](#) [2nd](#)
[AppController Interface in ApplicationController.h \(Listing 14.11\)](#)

product information system project (SOAP)

[-getProductInformation in ApplicationController.m \(Listing 15.12\)](#) [2nd](#) [3rd](#)
[-getProductInformation](#) **D** [Part 1 in ApplicationController.m \(Listing 15.13\)](#) [2nd](#)
[-getProductInformation](#) **D** [Part 2 in ApplicationController.m \(Listing 15.14\)](#) [2nd](#)
[-getProductInformation](#) **D** [Part 3 in ApplicationController.m \(Listing 15.15\)](#) [2nd](#) [3rd](#)
[AppController Interface in ApplicationController.h \(Listing 15.11\)](#)
[In the Beginning of product client.php \(Listing 15.8\)](#)
[In the End of product client.php \(Listing 15.10\)](#) [2nd](#)
[In the Middle of product client.php \(Listing 15.9\)](#) [2nd](#) [3rd](#) [4th](#)
[Initial Steps in product server.php \(Listing 15.2\)](#) [2nd](#)
[product client.php \(Listing 15.7\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#)
[product server.php \(Listing 15.1\)](#) [2nd](#) [3rd](#)
[The get product info Method in product server.php \(Listing 15.3\)](#) [2nd](#)
[The XML-Formatted Data Returned from product server.php \(Listing 15.5\)](#) [2nd](#)
[The XML-Formatted Data Sent to product server.php \(Listing 15.4\)](#) [2nd](#)

RadarWatcher program

[AppController.h](#)
[AppController.m](#) [2nd](#)
[main.m](#)
[MyDocument.h](#) [2nd](#) [3rd](#)
[MyDocument.m](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)
[PreferenceController.h](#)
[PreferenceController.m](#) [2nd](#)
[RadarView.h](#) [2nd](#)
[RadarView.m](#) [2nd](#) [3rd](#) [4th](#) [5th](#)

RadarWatcherX project

[AppController +initialize \(Listing 3.15\)](#) [2nd](#)
[AppController](#) **D** [applicationShouldOpenUntitledFile \(Listing 3.16\)](#) [2nd](#)
[AppController](#) **D** [showPreferencePanel \(Listing 3.17\)](#)
[application preferences](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#)
[Creating a Timer \(Listing 3.8\)](#)
[Main.m \(Listing 3.1\)](#) [2nd](#) [3rd](#)
[MyDocument -init \(Listing 3.3\)](#)

[MyDocument -writeFromFileOfType\(Listing 3.5\)](#)

[MyDocument -writeToFileOfType\(Listing 3.4\) 2nd 3rd](#)

[MyDocument !\[\]\(9b80616d1da42fc4431bc1cc919de00b_img.jpg\) refreshRadar method Image Checking \(Listing 3.11\) 2nd](#)

[MyDocument !\[\]\(b0ee107ab68edc75d1be1d156bcc1370_img.jpg\) refreshRadar method Image Loaded \(Listing 3.10\) 2nd](#)

[MyDocument !\[\]\(fe6bb28a2fa7487f6b8cf0824f8bd6e4_img.jpg\) refreshRadar method Image Loading \(Listing 3.9\)](#)

[MyDocument !\[\]\(164095c2009f19d94cfba343bcb17f0f_img.jpg\) updateUI \(Listing 3.6\) 2nd](#)

[NSMutableDictionary Keys \(Listing 3.2\)](#)

[PreferenceController.h \(Listing 3.14\)](#)

[PreferenceController.m \(Listing 3.18\) 2nd](#)

[RadarView -isColorInImageInRect\(Listing 3.13\) 2nd 3rd](#)

[RadarView color checker 2nd 3rd](#)

[RadarView instance variable 2nd](#)

[RadarView Mouse Overrides \(Listing 3.12\) 2nd](#)

[RadarView watch box 2nd](#)

[RadarWatcherX .rdrw Data File \(Listing 3.7\) 2nd](#)

services

[Reading Pasteboard Types \(Listing 7.7\) 2nd](#)

[Registering Pasteboard Types \(Listing 7.4\) 2nd](#)

[Validating Pasteboard Types \(Listing 7.5\) 2nd](#)

[Writing Pasteboard Types \(Listing 7.6\) 2nd](#)

Shared program

[MyCFCallPlugin.c 2nd](#)

[MyCFPluginInterface.h 2nd](#)

[ThePicker Interface in ThePicker.h \(Listing 11.1\) 2nd](#)

[ThePicker !\[\]\(ceabe4dca9bbf03392ec83f21df568f8_img.jpg\) currentMode Method in ThePicker.m \(Listing 11.5\)](#)

[ThePicker !\[\]\(7d62ddbe077f6e4c2a63caba57163986_img.jpg\) doRandomColorAction Method in ThePicker.m \(Listing 11.8\) 2nd](#)

[ThePicker !\[\]\(3e8e30d50bb920bde398d443d4b14acb_img.jpg\) initWithPickerMaskcolorPanel Method in ThePicker.m \(Listing 11.2\) 2nd](#)

[ThePicker !\[\]\(3f642ca29c4d42a0c92cdcd02ac839a1_img.jpg\) logText Method in ThePicker.m \(Listing 11.9\)](#)

[ThePicker !\[\]\(9932f6900486343afdea0f594e171290_img.jpg\) provideNewButtonImage Method in ThePicker.m \(Listing 11.3\)](#)

[ThePicker !\[\]\(86b0563b9e452bcee4a43de2a7f2ce07_img.jpg\) provideNewView Method in ThePicker.m \(Listing 11.7\)](#)

[ThePicker !\[\]\(772672fa91aa6d559ba8274ac8458c85_img.jpg\) setColor Method in ThePicker.m \(Listing 11.4\) 2nd](#)

[ThePicker !\[\]\(9bf26e7fa87cd3726c6a152bd02c489f_img.jpg\) supportsMode Method in ThePicker.m \(Listing 11.6\)](#)

SourceForge.net Web site

[Flat File DataBase \(FFDB\) PHP Library](#)

[PHP database resources](#)

[SplitForks utility \(command line\)](#)

[state machines](#)

[static libraries](#)

status items

[appearance of](#) [2nd](#) [3rd](#)

[Apple restrictions on use](#) [2nd](#) [3rd](#)

[function of](#) [2nd](#) [3rd](#)

MyStatusItem project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[customizing options](#)

[nib file \(Interface Builder\)](#)

[results overview](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#)

[Target settings \(Project Builder\)](#) [2nd](#)

[StepWise.com Web site](#)

[Stick Software Web site](#)

[symbol collisions \(Objective-C\)](#) [2nd](#)

system color panel

[Color Wheel color picker](#)

[folder location](#)

[RGB.CMYL color mode](#)

[System Preference Panes,](#) [See [preference panes](#)]

System Preferences application

Screen Effects pane

[Activation tab](#)

[Hot Corners tab](#)

[MyScreenSaver project](#) [2nd](#)

[Screen Effects tab](#)

[System Services,](#) [See [services](#)]

[[Team LiB](#)]

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

TCP protocol

[Mac OS X support](#)

terminal applications

files

[copying via wildcards](#)

functions

[man command](#)

MyTerminal project

[constructing \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#) [16th](#) [17th](#)

[18th](#)

[nib file \(Interface Builder\)](#)

[results overview](#) [2nd](#)

[source code \(Project Builder\)](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#) [15th](#)

[Target settings \(Project Builder\)](#)

MyThread project

[customizing options](#) [2nd](#)

[remote logons](#)

[rm command](#)

shell scripts

[executing](#)

[top command](#)

[UNIX](#)

[versus GUI interface](#)

[ThePicker.h code \(MyColorPicker program\)](#)

[ThePicker.m code \(MyColorPicker program\)](#) [2nd](#) [3rd](#)

third-party frameworks

[Omni Group](#)

[OmniAppKit](#) [2nd](#)

[OmniBase](#)

[OmniFoundation](#) [2nd](#)

[OmniHTML](#)

[OmniNetworking](#)

[OWF \(Omni Web Framework\)](#)

Thread Viewer

[window appearance](#)

threads

[function of 2nd](#)

MyThread project

[constructing \(Project Builder\) 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th](#)

[18th 19th 20th 21st 22nd](#)

[customizing options](#)

[nib file \(Project Builder\)](#)

[results overview 2nd](#)

[source code\(Project Builder\) 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th](#)

[18th 19th](#)

[Target settings \(Project Builder\)](#)

[overhead costs](#)

[parallel 2nd](#)

[priorities 2nd](#)

[serial 2nd](#)

[versus timers](#)

timers

[versus threads](#)

[toll-free bridging](#)

tools

Interface Builder

[development functions](#)

Project Builder

[Concurrent Versions System \(CVS\)](#)

[development functions](#)

[gcc3 debugger \(GNU Compiler Collection\)](#)

[gdb \(The GNU Debugger\) 2nd 3rd](#)

top command

[terminal applications](#)

[top utility \(command line\)](#)

[TripleSoft.com Web site](#)

[\[Team LiB \]](#)

[\[Team LIB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

UDP protocol

[Mac OS X support](#)

UFS file system

[Mac OS X support](#)

[Universally Unique Identifiers](#), [\[See UUIDs\]](#)

UNIX

applications

[mac OS X default installations](#)

[benefits](#)

[BSD \(Berkeley Software Distribution\)](#)

[historical background](#)

[Process Viewer](#)

servers

[stability of](#)

[terminal application](#)

versions

[AT&T](#)

[BSD](#)

[Linux](#)

[UnRezWack utility \(command line\)](#)

updating

[application dock images](#) [2nd](#)

[application Dock menu](#) [2nd](#)

URLs

[AppleScript execution](#)

[opening \(code snippet\)](#)

[USB Prober](#)

usernames

[displaying \(code snippet example\)](#)

users

application defaults

[defaults command-line utility](#)

[utilities](#) [\[See also command line utilities\]](#)

[Apple Help Indexing Tool](#)

[FileMerge](#)

[icns Browser](#)

[IconComposer](#)

[IORegistryExplorer](#)

[JavaBrowser](#)

[MallocDebug](#)

[ObjectAlloc](#)

[OpenGL Info](#)

[OpenGL Profiler](#)

[OpenGL Shader Builder](#)

[PackageMaker](#)

[PEFViewer](#)

[Pixie](#)

[Property List Editor](#)

[Quartz Debug](#)

[Sampler](#)

[Thread Viewer](#)

[USB Prober](#)

UUIDCreator application

[number generator](#)

UUIDs

[\(Universally Unique Identifiers\)](#)

[ASCII format](#)

[HEX format](#)

[UUIDCreator application](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

validating

pasteboard types

[services](#) [2nd](#)

[vmmap utility \(command line\)](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

W3C Web site

[SOAP specification resources](#)

Web sites

[Adobe Photoshop 2nd](#)

[Apple Developer](#)

[localization resources](#)

[Apple Developer Connection](#)

Apple Developers

[Objective-C resources](#)

[Cocoa Browser](#)

[CocoaDev.com](#)

[CocoaDevCentral.com](#)

[Icon Factory](#)

[Mac OS X Hints](#)

[Omni Group](#)

[OmniGroup.com](#)

[OpenGL.org](#)

SourceForge.net

[Flat File DataBase \(FFDB\) PHP Library](#)

[PHP database resources](#)

[StepWise.com](#)

[Stick Software](#)

[TripleSoft.com](#)

W3C

[SOAP specification resources](#)

[XML-RPC.org](#)

[WebCore.framework](#)

WebDAV file system

[Mac OS X support](#)

writing

pasteboard types

[services 2nd](#)

[WSMakeStubs utility \(command line\)](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

X11 technology

[Darwin implementation](#)

XML

text files

[viewing \(Property List Editor\)](#)

XML-RPC

[data-based client/server architecture](#)

[development of](#)

[features](#) [2nd](#)

product information project

[Cocoa client](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#) [11th](#) [12th](#) [13th](#) [14th](#)

[PHP client](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

[PHP server](#) [2nd](#) [3rd](#) [4th](#) [5th](#) [6th](#) [7th](#) [8th](#) [9th](#) [10th](#)

[results overview](#) [2nd](#) [3rd](#)

[Web site](#)

XML-RPC protocol

[Mac OS X support](#)

[\[Team LiB \]](#)

[\[Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[Yahoo.plist code \(plistPlugin program\)](#)

[\[Team LiB \]](#)