Fritz Anderson

Xcode 5
Start to Finish

iOS and OS X Development

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Xcode 5
Start to Finish

Developer's Library Series

Visit **developers-library.com** for a complete list of available products

---

The **Developer's Library Series** from Addison-Wesley provides practicing programmers with unique, high-quality references and tutorials on the latest programming languages and technologies they use in their daily work. All books in the Developer's Library are written by expert technology practitioners who are exceptionally skilled at organizing and presenting information in a way that's useful for other programmers.

Developer's Library books cover a wide range of topics, from open-source programming languages and databases, Linux programming, Microsoft, and Java, to Web development, social networking platforms, Mac/iPhone programming, and Android programming.

PEARSON

Addison-Wesley | Cisco Press | EXAM/CRAM | IBM Press | QUE | PRENTICE HALL | SAMS | Safari Books Online

Xcode 5 Start to Finish

iOS and OS X Development

Fritz Anderson

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data
Anderson, Fritz.
Xcode 5 start to finish : iOS and OS X development / Fritz Anderson.
pages cm
Includes index.
ISBN 978-0-321-96720-6 (pbk. : alk. paper)—ISBN 0-321-96720-8 (pbk. : alk.paper)
1. Mac OS. 2. iOS (Electronic resource) 3. Macintosh (Computer)—Programming. 4. iPhone (Smartphone)—Programming. 5. Application software—Development. I. Title.
QA76.774.M33A53 2014
005.3'82—dc23
2014000229

ISBN-13: 978-0-321-96720-6
ISBN-10: 0-321-96720-8

Text printed in the United States on recycled paper at Edwards Brothers Malloy in Ann Arbor, Michigan.
First printing, May 2014

Editor-in-Chief
Mark L. Taub

Senior Acquisitions Editor
Trina MacDonald

Senior Development Editor
Chris Zahn

Managing Editor
John Fuller

Full-Service Production Manager
Julie B. Nahil

Copy Editor
Stephanie Geels

Indexer
Ted Laux

Proofreader
Andrea Fox

Technical Reviewers
Duncan Champney
Chuck Ross
Dan Wood

Editorial Assistant
Olivia Basegio

Cover Designer
Chuti Prasertsith

Compositor
LaurelTech

❖

For Magdalen Jeanette Anderson (1952–2013),
the mother of my children

❖

This page intentionally left blank

Contents at a Glance

Contents

Acknowledgments

Only part of the effort that went into putting *Xcode 5 Start to Finish* into your hands was spent at a text editor. I am indebted to those without whom this book could not have been made. Many of them appear in the formal production credits; they deserve better-than-formal thanks.

Trina MacDonald guided me through the from-scratch planning process that a new Xcode book required, and advocated for it when the need was not obvious.

Olivia Basegio made sure the contracts, correspondence, (and advance payments!) all got through. She herded the reviewers through their work while the book was still pieces lying on the ground.

The reviewers—Duncan Champney, Chuck Ross, and Dan Wood—were friends to the book even through the burden of trying to make sense of a work that spiraled, rather than marched, to its conclusion. They saved me much embarrassment, and made this a much better work than it started. Errors remain. Some are intentional, some not; they are all my own.

Julie Nahil, the production manager, and Stephanie Geels, the copy editor, made it exceptionally easy to give you a book that is as close as it can be to what I meant it to be. The process was never smoother.

A full-time day job is not an author's best friend (except for the part about paying the rent), but Emerging Technologies, in the IT Services department of The University of Chicago, was a friend to me. My boss, Alan Takaoka, got me three-day weekends while I wrote, even when I ran over by a couple of weeks. I promised to keep all my Monday meetings and deadlines while I worked on the book, but my colleague, Cornelia Bailey, made the deadlines on our projects disappear.

Bess and Kate bore more than daughters should of my doubts and frustrations, and were simply confident that I would do fine—which was all they needed to do.

This page intentionally left blank

About the Author

Fritz Anderson has been writing software, books, and articles for Apple platforms since 1984. He has worked for research and development firms, consulting practices, and freelance. He was admitted to the Indiana bar, but thought better of it. He is now a senior iOS developer for the Emerging Technologies and Communications division at The University of Chicago. He has two daughters.

This page intentionally left blank

Introduction

Welcome to *Xcode 5 Start to Finish*! This book will show you how to use Apple's integrated development environment to make great products with the least effort.

Xcode 5 is the descendant of a family of development tools dating back 20 years to NeXT's ProjectBuilder. It started as a text editor, a user-interface designer, and a front end for Unix development tools. It has become a sophisticated system for building applications and system software, with a multitude of features that leverage a comprehensive indexing system and subtle incremental parser to help you assemble the right code for your project, and get it right the first time.

That much power can be intimidating. My aim in *Xcode 5 Start to Finish* is to demystify Xcode, giving you a gradual tour through examples that show you how it is used day to day. Don't let the gentle approach fool you: This book will lead you through the full, best-practices workflow of development with Xcode 5. There are no "advanced topics" here—I'll show you version control and unit testing in their proper places in the development process.

How This Book Is Organized

First, a word on my overall plan for *Xcode 5 Start to Finish*. This is a book about developer tools. If it teaches you something about how to use the Cocoa frameworks, or something about programming, that's fine, but that's incidental to showing you the Xcode workflow. There are many excellent books and other resources for learning the frameworks; you'll find many of them listed in Appendix B, "Resources."

Every tour needs a pathway, and every lesson needs a story. The first three parts of this book demonstrate Xcode through three applications—a command-line tool, an iOS app, and an OS X application—that calculate and display some statistics in American football. None of the apps are very useful; the graphical apps run almost entirely on sample data. But they demand enough of the development tools to give you a solid insight into how to use them.

The full code for the example programs is available online from `informit.com/9780321967206` (register your book to gain access to the code). In the interest of space, I'll be showing only excerpts.

Xcode supports some technologies, like Core Data and OS X bindings, that are *not* for beginners. *Xcode 5 Start to Finish* dives straight into those techniques, ignoring conceptually simpler approaches, so I can demonstrate how Xcode works. Other "advanced" techniques, like unit testing and version control, appear at the points where best practices require them. Again, I'll be showing you the workflow as Xcode supports it.

I'm using applications for iOS and OS X as examples, but read both Parts II and III, even if you're only interested in one platform. The applications are only *stories*; the techniques apply to both platforms.

First Steps

In Part I, I'll take you from installing Xcode and running your first project through basic debugging skills. You'll work through a small command-line application. The application may be simple, but you'll learn foundational skills you'll need before adding the complexity of graphical apps.

- **Chapter 1, Getting Xcode**—Some things to consider before you download Xcode 5; two ways to download and install it.
- **Chapter 2, Kicking the Tires**—Your first look at Xcode, setting up a trivial project and running it.
- **Chapter 3, Simple Workflow and Passive Debugging**—Write, build, and run a simple application, and respond to a crash.
- **Chapter 4, Active Debugging**—Take charge of debugging by setting breakpoints and tracing through the program. I'll show you how to organize your workspace.
- **Chapter 5, Compilation**—A pause for a description of the process of building an application.
- **Chapter 6, Adding a Library Target**—Add a library target to a project, and learn how to build a product from multiple targets.
- **Chapter 7, Version Control**—Why source control is important, and how to take advantage of Xcode's built-in support for versioning through Git and Subversion.

The Life Cycle of an iOS Application

Part II tells the story of a small iPhone app, and how to use Apple's developer tools to build it. It introduces you to the graphical editor for user interfaces, and shows how to profile an app to optimize its speed and memory burden.

- **Chapter 8, Starting an iOS Application**—You'll start by creating an iOS project, and learn the Model-View-Controller design at the core of Cocoa on iOS and OS X alike.
- **Chapter 9, An iOS Application: Model**—Design a Core Data schema and supplement it with your own code.
- **Chapter 10, An iOS Application: Controller**—Create a controller to link your model to the on-screen views. On the way, I'll tell you about refactoring, and Xcode's continual error-checking.
- **Chapter 11, Building a New View**—Design the user-interface views for your app with the integrated Interface Builder, and take advantage of source-code completion.

- **Chapter 12, Autolayout in a New View**—In Xcode 5, autolayout is more about getting things done than fighting the tools. Learn how to make Cocoa layout do what you want.
- **Chapter 13, Adding Table Cells**—While adding an in-screen component to your app, you'll debug memory management, and control how Xcode builds, runs, and tests your apps through the Scheme editor.
- **Chapter 14, Adding an Editor**—Add an editor view, and get deep into Storyboard.
- **Chapter 15, Unit Testing**—Unit testing speeds development and makes your apps more reliable. I'll show you how Xcode supports it as a first-class part of the development process.
- **Chapter 16, Measurement and Analysis**—Use Instruments to track down performance and memory bugs.
- **Chapter 17, Provisioning**—Behind the scenes, the process of getting Apple's permission to put apps on devices is complicated and temperamental. I'll show you how Xcode saves you from most of the pain, and give you a few tips on how to get out if it backs you into a corner.

Xcode for Mac OS X

Part III shifts focus to OS X development. Some concepts are more important to OS X than iOS, but you'll be learning techniques you can use regardless of your platform.

- **Chapter 18, Starting an OS X Application**—Carrying iOS components over to OS X; what the responder chain is, and how Interface Builder makes it easy to take advantage of it.
- **Chapter 19, Bindings: Wiring an OS X Application**—As you build a popover window, you'll use OS X bindings to simplify the link between your data and the screen. You'll also encounter autosizing, a legacy technique for laying out view hierarchies.
- **Chapter 20, A Custom View for OS X**—Add a custom view to your app, and see how Interface Builder can lay it out and configure it, even though it's not a standard Cocoa component.
- **Chapter 21, Localization**—How you can translate your Mac and iOS apps into other languages.
- **Chapter 22, Bundles and Packages**—You'll master the fundamental structure of most Mac and iOS products, and how both platforms use the `Info.plist` file to fit apps into the operating system.
- **Chapter 23, Frameworks**—Package and share a complete subprogram you can incorporate into any OS X application.
- **Chapter 24, Property Lists**—Learn the basic JSON-like file type for storing data in both OS X and iOS.

Xcode Tasks

By this point in the book, you'll have a foundation for digging into the details of the Xcode toolset. Part IV moves on to topics that deserve a more concentrated treatment than Parts II and III.

- **Chapter 25, Documentation in Xcode**—How Xcode gives you both immediate help on API, and browsable details on the concepts of Cocoa development. Find out how you can add your own documentation to the system.
- **Chapter 26, The Xcode Build System**—I'll show you the fundamental rules and tools behind how Xcode goes from source files to executable products.
- **Chapter 27, Instruments**—Using Apple's timeline profiler, you can go beyond basic performance and memory diagnostics to a comprehensive look at how your program uses its resources.
- **Chapter 28, Debugging**—How to use breakpoint actions and conditions to eliminate in-code diagnostics. You'll also find a tutorial on the `lldb` debugger command set, for even closer control over your code.
- **Chapter 29, Continuous Integration**—Mavericks Server complements Xcode 5 with a sleek continuous-integration system that can synthesize your code analysis, perform cross-platform unit tests, and generate product archives. I'll show you how to get started, and how to put it to best use.
- **Chapter 30, Snippets**—A roundup of tips, traps, and features to help you get the most from the developer tools.

Appendixes

The appendixes in Part V contain references to help you master the build system, and find help and support.

- **Appendix A, Some Build Variables**—The most important configuration and environment variables from Xcode's build system.
- **Appendix B, Resources**—A compendium of books, tools, and Internet resources to support your development efforts.

About Versions

This book was finished in the fall of 2013, shortly after Apple released iOS 7, OS X Mavericks, and Xcode 5 to the public. *Xcode 5 Start to Finish* is written to the first-bugfix versions of all three.

About the Code

Xcode 5 Start to Finish has many examples of executable code—it's about a system for creating code and running it. My goal is to teach *workflow.* What the code itself does is

practically incidental. In particular, be aware: **Much of the code in this book will not run as initially presented.** *Xcode 5 Start to Finish* is about the development process, most of which (it seems) entails prosecuting and fixing bugs. You can't learn the workflow unless you learn how to respond to bugs.

So I'll be giving you buggy code. You may find it painful to read, and if you try to run it, it will be painful to run. Trust me: It's for a reason.

Also, sample code for this book is available at `informit.com/title/9780321967206` (register your book to gain access to the code). You'll find archives of the projects in this book as they stand at the end of each chapter. With very few exceptions—I'll make them very clear—if you want the project as it stands at the *start* of a chapter, you should use the archive for the *end* of the previous chapter.

The chapter archives do not include version-control metadata. If you are following along with the examples, and using Git (or Subversion) for your work, copy the changes into your own working directory. If you replace your directory with a sample directory, you'll lose your version history.

Conventions

This book observes a number of typographic and verbal conventions.

- Human-interface elements, such as menu items and button labels, are shown **like this**.
- File names and programming constructs are shown `like this`. This will sometimes get tricky as when I refer to the product of the "Hello World" *project* (plain text, because it's just a noun) as the *file* `Hello World`.
- Text that you type in will be shown **`like this`**.
- When I introduce a new term, I'll call it out *like this*.

I'll have you do some command-line work in the Terminal. Some of the content will be wider than this page, so I'll follow the convention of breaking input lines with backslashes (\) at the ends. I'll break long output lines simply by splitting them, and indenting the continuations. When that output includes long file paths, I'll replace components with ellipses (. . .), leaving the significant parts.

For many, many years the Macintosh had a one-button mouse. (Don't laugh—most purchasers didn't know what a mouse *was*; try pushing the wrong button on an old Mac mouse.) Now it has four ways to effect an alternate mouse click: You can use the right button on an actual mouse; you can hold down the Control key and make an ordinary click; you can hold down two fingers while clicking on a multi-touch trackpad (increasingly common even on desktop Macs); or you can tap at a designated corner of a multi-touch trackpad. And there are more variations available through System Preferences. Unless the distinction really matters, I'm simply going to call it a "right-click" and let you sort it out for yourself.

This page intentionally left blank

2

Kicking the Tires

Now you have Xcode. It's time to start it up and see what it looks like.

Starting Xcode

You'll find Xcode in the `/Applications` directory, just like any application. You'll be using it constantly, so you'll want to keep it in the Dock at the bottom of your main screen. Drag Xcode to the Dock—take care to drop it *between* icons, and not *on* one.

Now click on the Xcode icon. It bounces to show Xcode is being launched. The first time you run any of Apple's developer tools—even through the command line—you'll be asked to read and accept a license agreement for the tools and SDKs. It's no different from any other click-through license process.

Next, Xcode will ask you for permission to install "additional components" it needs. Permit it, and present an administrator's credentials. Those components overlap the iTunes frameworks, so you may be asked to close iTunes.

Once the progress window clears, you are greeted with the "Welcome to Xcode" window (see Figure 2.1).

If this is the first time you've ever run Xcode, the table on the right will be empty ("No Recent Projects"); as you accumulate projects, the table will contain references to them, so you have a quick way to get back to your work. When you accumulate projects in this list, you'll be able to select one, but Xcode doesn't reveal any way to open it. The trick is to double-click the item, or press the Return key.

You have three other options:

- **Create a new Xcode project.** This is obvious enough; it's how you'd start work on a new product. You're about to do this, but hold off for the moment. You could also select **File →New →New Project. . .** (⇧⌘**N**).
- **Check out an existing project.** Xcode recognizes that source control management is essential to even the most trivial of projects. Your development effort might start not with your own work, but with collaborative work pulled in from a source repository. Use this link to get started.

Figure 2.1 When you launch Xcode, it displays a "Welcome" window with options for creating a new project, reopening a recent one, or fetching a project from a source-control repository.

- **Open Other. . .** (at the bottom of the "recents" list). This will get you the standard get-file dialog so you can select any Xcode project file you want. You can do the same thing with the **File →Open. . .** (⌘**O**) command.

If you need to get back to the Welcome window, select **Window →Welcome to Xcode** (⇧⌘**1**). If you're tired of seeing this window, uncheck **Show this window when Xcode launches**.

> **Note**
>
> "Show this window when Xcode launches" is not quite accurate. If you had a project open when you last quit Xcode, it will reopen automatically when you start it up again, and the Welcome window won't appear.

Hello World

Just to get oriented, I'm going to start with the simplest imaginable example project—so simple, you won't have to do much coding at all.

A New Project

Click the **Create a new Xcode project** link. Xcode opens an empty Workspace window, and drops the New Project assistant sheet in front of it (see Figure 2.2). Select **OS X →Application** from the list at left, and then the **Command Line Tool** template from the icons that appear at right. Click **Next**.

The next panel (Figure 2.3) asks for the name of the project and the kind of command-line tool you want.

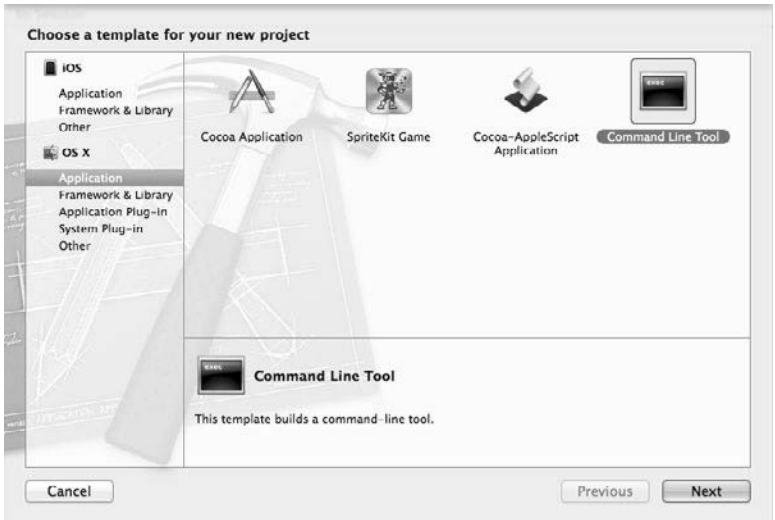

Figure 2.2 The New Project assistant leads you through the creation of an Xcode project. First, you specify the kind of product you want to produce.

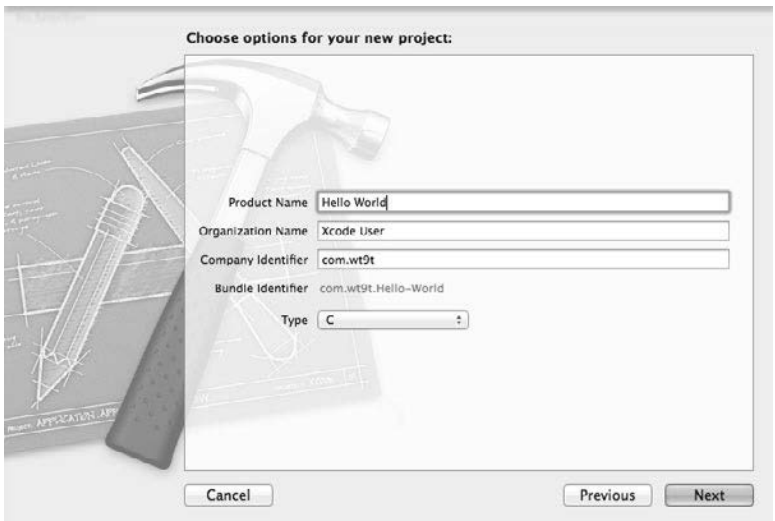

Figure 2.3 The **Options** panel of the New Project assistant lets you identify the product and what support it needs from system libraries.

1. Type **`Hello World`** in the **Product Name** field. This will be used as the name of the project and its principal product.
2. Xcode should have filled the **Organization Name** in for you, from your "me" card in the Address Book. If you listed a company for yourself, that's what will be in the field; otherwise, it's your personal name. Xcode will use this as the name of the copyright holder for all new files.
3. Virtually all executable objects in the OS X and iOS world have unique reverse-DNS-style identifiers that are used to uniquely identify them. The **Company Identifier** is the leading portion of those reverse-DNS names, to be used by every product of this project. For this example, I used `com.wt9t`.
4. By default, Xcode infers the unique **Bundle Identifier** from the company identifier and the name of the product. You'll see later how to change this if you have to.
5. The **Type** popup prompts Xcode on how to fill in the system libraries the tool will use. This is just a plain old C program, with no need for C++ or Apple-specific support, so choose **C**.

Finally, a put-file sheet appears, so you can select a directory to put the project into. For this example, select your Desktop. One more thing—*uncheck* the box labeled **Create local git repository for this project**. Source control (Chapter 7, "Version Control") is a Good Thing, but let's not deal with it in this trivial example. Click **Create**.

If you look on your Desktop, you'll find that Xcode has created a folder named `Hello World`. The project name you specified is used in several places.

- It's the name of the project *directory* that contains your project files.
- It's the name of the project *document* (`Hello World.xcodeproj`) itself.
- It's the name of the *product*—in this case a command-line tool named `Hello World`.
- It's the name of the *target* that builds the product. I'll get into the concept of a target later; for now, think of it as the set of files that go into a product, and a specification of how it is built.
- It's the name of the *target's* directory inside the *project's* directory.
- Suitably modified, it's the name of the `man`-file template for the tool (`Hello_World.1`).

When you've made your choices, Xcode unmasks the workspace for the Hello World project (Figure 2.4). Don't look at it too closely just yet. Xcode starts you off with a view of the settings that control how `Hello World` is to be built. This is useful information, but for now, it's just details.

More interesting is the program code itself. The left column of the window is called the *Navigator area*. Find `main.c` in the list, and click it (see Figure 2.5). The Editor area, which fills most of the window, now displays the contents of `main.c`. This is the code for the canonical simplest-possible program, known as "Hello, World."

The Navigator area displays many different things in the course of development—it's not just a file listing. It can display analyses, searches, and build logs. Which list you see often depends on what Xcode wants to show you; you can make the choice yourself by

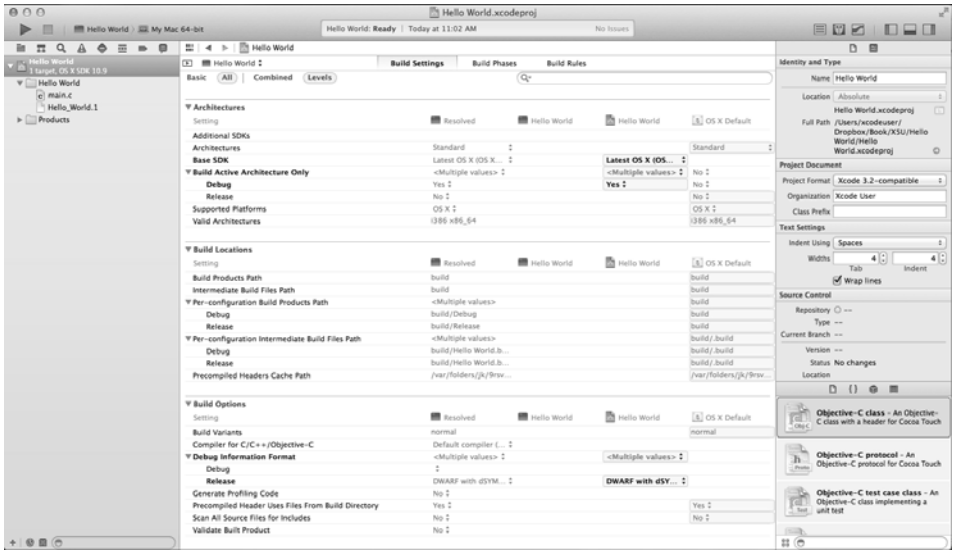

Figure 2.4 Once set up, the Hello World project window fills in with a list of source files and a display of the options that control how the application will be built.

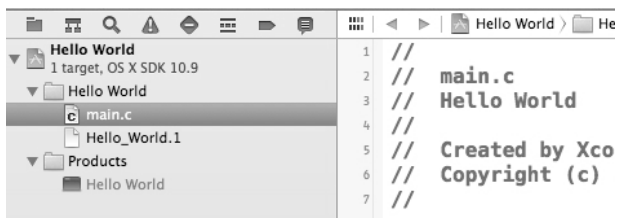

Figure 2.5 Clicking the name of an editable file in the Project navigator displays its contents in the Editor area.

clicking the tiny icons in the bar at the top of the Navigator area. Hovering the mouse pointer over them will show you the names of the various views.

As this book goes on, you'll meet all of them. For now, you care only about the "Project" navigator, the file list Xcode starts you out with. Feel free to click the other icons, but to keep up with this example, be sure to return to the Project navigator, as shown in Figure 2.5.

Quieting Xcode Down

But first.

Xcode is a toolset that contains everything its creators could think of to provide a powerful, helpful environment for writing iOS and OS X applications. Often, you barely need to begin a task, and Xcode will offer to finish it for you. It will usually be right. I use these features all the time. I recommend them.

You're going to turn them all off.

Automatic completions and indentations and code decorations and code fixes are great, once you know what's going on, but an automaton that snatches your work out of your hands, however helpfully, is straight out of *The Sorcerer's Apprentice*. Better to start with what *you* want to do; once you're confident of what that is, then you have the discretion and control to direct Xcode as it helps you.

So you're going to damp Xcode down a bit. You'll do all of this in Xcode's Preferences window, which you can summon with **Xcode →Preferences. . .** (**⌘ comma**). The Preferences window is divided into panels, which you select with the icons at the top of the window.

To start, make sure the **General** panel is visible. Under **Issues**, uncheck **Show live issues**.

Next, select the **Text Editing** panel, which has two tabs. Select the **Editing** tab, and uncheck **Show: Code folding ribbon**, and all the options under **Code completion:**.

In the **Indentation** tab, turn off **Line wrapping: Wrap lines to editor width** and the **Syntax-aware indenting** section.

Now Xcode will mostly stay out of your way as you explore.

Building and Running

The program in `main.c` would run as is, but we have to trick Xcode into keeping its output on the screen long enough to see it. Add a few lines after the `printf` call so it looks like this:

```
int main(int argc, const char * argv[])
{

    // insert code here...
    printf("Hello, World!\n");

    /****************************************
     *  Pause, so the console doesn't disappear
     ****************************************/
    char    dummy[128];
    fgets(dummy, sizeof(dummy), stdin);

    return 0;
}
```

Now we can run it. Select **Product →Run** (**⌘R**).

In the ordinary course, Xcode would then build and run `Hello World`. However, if this is the first time you've run any application, there is a security problem: Running an app from Xcode puts it under the observation of a debugger, which will have access to the internal data and running state of the app. Crossing process boundaries like that is technically a security breach, and you have to authorize it.

Xcode posts an alert, **Enable Developer Mode on this Mac?**. It explains that you could be asked for an administrator's password every time you run the debugger ("**Developer Tools Access needs to take control of another process. . .** "), or, with Developer Mode, you could do the authorization once and forget about it. Click **Enable**, enter an administrator's credentials, and carry on.

> **Note**
>
> You can turn Developer Mode off, or on again, from the Devices organizer (⇧⌘**2**). Select your Mac and click the button **Disable Developer Mode**.

With authorization taken care of, a heads-up window ("bezel") appears almost instantly, to tell you "Build Succeeded." (If Xcode is in the background, a notification banner will appear saying the build succeeded, and identifying the project and product involved.)

So. What happened?

`Hello World` is a console application; it just writes out some text without putting up any windows. Xcode captures the console of the apps it runs in the Debug area, which popped into view when you ran the program (Figure 2.6). The Debug area includes a console view on the right. It says `Hello, World!` (Figure 2.7).

Click in the console to make it ready for text input, and press the Return key. `Hello World` exits, and the Debug area closes.

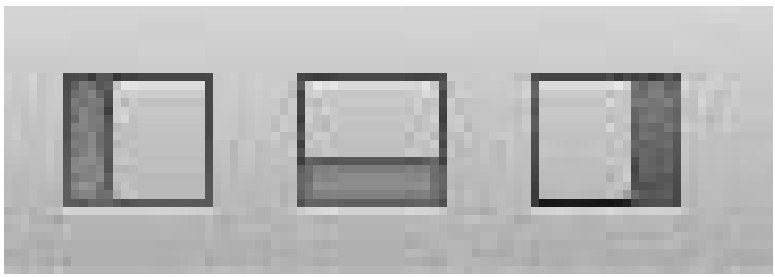

Figure 2.6 The **View** selector in the toolbar shows and hides the Navigator, Debug, and Utility areas (left to right) of the project window. Clicking a button so it is highlighted exposes the corresponding area. Here, the Navigator and Debug areas are selected.

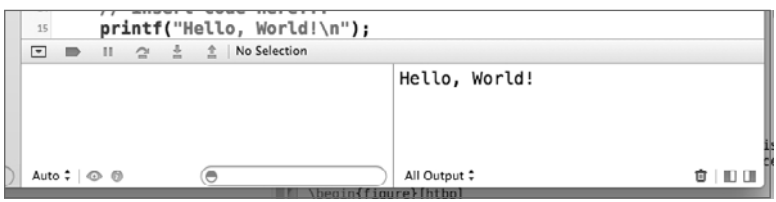

Figure 2.7 Opening the Debug area after running `Hello World` shows the eponymous output.

> **Note**
>
> If the Debug area didn't hide itself as soon as an application terminated, we wouldn't have had to add that `fgets()` call. That's easy to change; see the "Behaviors" section of Chapter 4, "Active Debugging."

The Real Thing

What Xcode just produced for you is a real, executable application, not a simulation. To prove it, open the Terminal application (you'll find it at `/Applications/Utilities/Terminal`, and you'd be well advised to add Terminal to your Dock). In Xcode, find the `Hello World` product in the Project navigator by clicking the disclosure triangle next to the Products folder icon. Drag the `Hello World` icon into the Terminal window, switch to Terminal, and press the Return key. (The path to a file deep in a directory for build products is remarkably long, but Terminal takes care of the necessary escaping.) "Hello, World!" appears.

If you want access to the executable file itself, select it in the Project navigator, then **File → Show in Finder**—also available in the contextual menu you get by right-clicking the `Hello World` icon. A window will open in the Finder showing you the file.

You're done! You can close the Workspace window (**File → Close Project**, ⌥⌘**W**) or quit Xcode entirely (**Xcode → Quit Xcode**, ⌘**Q**).

Getting Rid of It

There is nothing magical about an Xcode project. It's just a directory on your hard drive. If you don't want it any more, close the project, select its enclosing folder in the Finder, and drag it to the Trash. It's gone. It won't even show up in the Recents list in the Welcome to Xcode window, or in the **File → Open Recent** menu.

That's it.

Okay, yes, the build products of the project will still stick around in a warren of directories inside `~/Library/Developer/Xcode/DerivedData`. They aren't many or large in this case, but there's a principle involved.

If you want them gone, the best way is to close the project window, open the Organizer window (**Window → Organizer**, ⇧⌘**2**), select the **Projects** panel, select the "Hello World" project, press Delete, and confirm the deletion in the ensuing alert sheet. All trace of the build products is gone.

Summary

In this chapter, you had your first look at Xcode, and you discovered that it doesn't bite. You saw how to create a simple project, one you didn't even have to edit. You saw what happens when you run a project in Xcode, how to close a project and quit Xcode, and at last how to get rid of the project entirely.

Next, we'll start doing some real work.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z