Après Code: Distributing Your Application

This has been quite a journey, hasn’t it? You’ve learned a lot in the pages of this book; enough, we hope, to give you the tools you need to become a great Objective-C programmer. Let’s take a look back over everything you’ve seen so far.

- You were introduced to object-oriented programming following the Smalltalk model and how it differs from the Simula model adopted by languages such as C++ and Java.
- You saw the powerful and dynamic nature of messaging and learned to use blocks to pass around functions like first-class objects.
- You saw how blocks enable easy asynchronous programming techniques by capturing locally scoped variables on your behalf, saving you a lot of work maintaining state information.
- The flexible and powerful classes that make up the Foundation and Application kits were opened wide, bringing with them commonly used programming models such as delegation and dynamic programming.
- Many new language features were introduced, such as synthesized properties, collection subscripting, and object literals, along with more common elements like exceptions, synchronization, and interface protocols.
- You saw the many ways of working with the filesystem and you learned their relative strengths; you also learned to detect the common pitfalls of working with filesystem data and the facilities provided to best work around them, such as file coordination and bookmarks.
- You saw the many different ways of communicating across local- and wide-area networks, and you learned the properties and capabilities of the different protocols and transport media in use.
The concepts of view hierarchies and window management have been laid out and described in detail, and you saw how to use Cocoa Auto Layout to build fluidly responsive user interfaces.

You saw how the Cocoa drawing model works and how user input and layout events are passed through the view hierarchy.

Core Data was brought into the picture in grand fashion, showing how you can define your data model in an object-oriented yet code-free fashion, with the framework providing lots of support for you.

You saw how to use Cocoa Bindings to fit your data into a user interface in an almost entirely code-free fashion.

You worked with the application sandbox and used XPC services to break the core functionality of an application into properly isolated components with secure, well-defined interfaces.

You saw firsthand the benefits of dropping down a layer to the C-based Grand Dispatch APIs for their easy concurrency model and data- and throughput-handling facilities.

There are still a few steps between you and your first application, however. First of all, what about the iPhone and iPad? iOS uses the same Foundation framework as OS X, the same Grand Central Dispatch libraries, the same media libraries, and more. Then there’s the issue of getting your application out there to the rest of the world, either through the iOS, the Mac App Store, or distributing it through your own web site. We’ll give you an overview of all these topics.

Whither iOS?

Mac and iOS development is done using the same language: Objective-C. They even use a lot of the same frameworks and libraries under the hood. Almost all of the Foundation library can be used in either platform, including NSString, NSArray, NSDictionary, NSURLConnection, and so on. Even some platform-specific classes, like NSImage on OS X and UIImage on iOS, behave so similarly that it’s common to find yourself typing the prefix of the wrong class when you’re doing development on both platforms.

OS X developers have been using AppKit and Cocoa to write their apps for the Mac. In the time since these frameworks were written, they have accumulated cruft—same as any code. When Apple launched the iOS SDK in 2008, they took a fresh look at the user interface libraries and released the UIKit framework as part of Cocoa Touch.

The new frameworks benefit from the experience Apple gained writing AppKit and Cocoa. Apple used decades of writing, maintaining, and deprecating libraries to release UIKit and Cocoa Touch as very solid, mature frameworks, while leaving behind the mechanics dictated by two-decade-old technology.

While Cocoa has always used the Model-View-Controller paradigm, it’s certainly possible to write apps on OS X without following the principles of MVC. When they released Cocoa Touch, Apple formalized the relationship between models, views, and controllers by providing