Advanced iOS 4 Programming: Developing Mobile Applications for Apple iPhone, iPad, and iPod touch

Maher Ali, PhD

Bell Labs, Alcatel-Lucent

A John Wiley and Sons, Ltd., Publication

This edition first published 2010

Registered office
John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

Editorial office
John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com/wiley-blackwell.

ISBN 978-0-470-97123-9 (paperback),
ISBN 978-0-470-97144-4 (ebk),
ISBN 978-0-470-97165-9 (ebk),
ISBN 978-0-470-97954-9 (ebk)

A catalogue record for this book is available from the British Library.

Typeset in 10/12 Times by Laserwords Private Limited, Chennai, India
Printed in the United States of America

Contents

1

Getting Started

This chapter serves as a quick introduction to the tools bundled with the iOS SDK. It also shows you basic development steps used on the iOS operating system that include coding, user interface (UI) design, and debugging. You do not have to understand everything in this chapter as we will go over these concepts throughout the book. What you need to get from this chapter is a feeling for iOS development using XCode.

We start with some basics of the XCode IDE in Section 1.1. Next, Section 1.2 talks about the UI design tool Interface Builder. After that, we show you how to use the built-in debugger in XCode in Section 1.3. Next, Section 1.4 shows you different sources of information for obtaining additional help. Finally, we summarize the chapter in Section 1.5.

1.1 iOS SDK and IDE Basics

In this section, we walk you through the process of creating your first iPhone application. But first, you need to obtain the iOS SDK and install it on your Intel-based Mac.

1.1.1 Obtaining and installing the SDK

Obtaining and installing the iOS SDK is easy. Just follow these steps:

1. Get your iPhone developer Apple ID and password from:
   `http://developer.apple.com/iphone/`
2. Download the latest iOS SDK from the site mentioned above.
3. Install the iOS SDK on your Mac.

Now, you're ready to create your first project — read on!

1.1.2 Creating a project

Let's use XCode to create an iOS project targeting the iPhone device. First, locate XCode and launch it. You can use `Spotlight` to find it or you can navigate to `/Developer/Applications/XCode`.

XCode is the central application for writing, designing, debugging, and deploying your iOS applications. You will use it a lot, so go ahead and add it to the `Dock`.

From XCode, select `File` → `New Project`. You should see a window, similar to the one shown in Figure 1.1, asking you for the type of project you want to create. Choose the default and create a window-based application. This is the most generic type of iPhone project and the one that can be customized for different needs.

Figure 1.1 Choosing a window-based application in the project creation process.

Click `Choose`, enter the name of your project (here, we're using `My Project`), and click `Save`. A new directory is created with the name you entered, and several files are generated for you. You should now see the newly created iPhone project as shown in Figure 1.2.

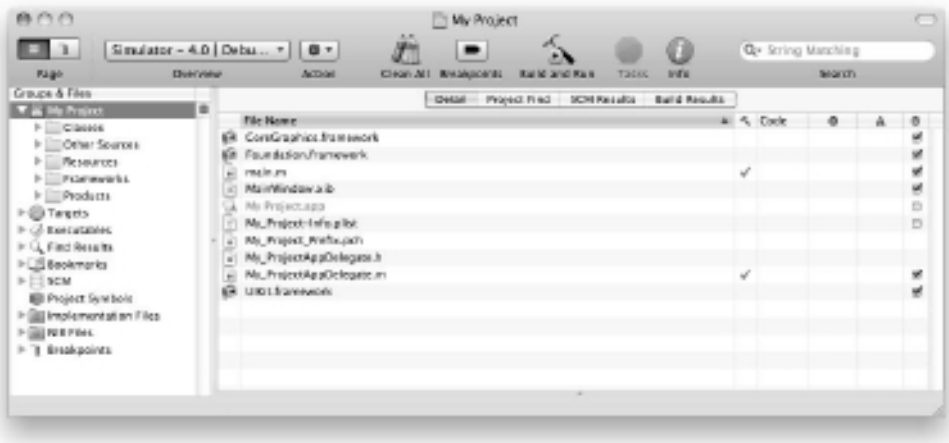

Figure 1.2 A newly created iPhone project in XCode.

1.1.3 Familiarizing yourself with the IDE

As you can see from Figure 1.2, the main window is divided into several areas. On the top, you will find the toolbar (Figure 1.3). The toolbar provides quick access to common tasks. It is fully configurable; you can add and remove tasks as you want. To customize the toolbar, Control-click it and choose `Customize Toolbar`. A window with a set of items will be shown so you can drag your favorite task on the toolbar. Click `Done` when you're finished. To remove an item, Control-click on it and choose `Remove Item`.

Figure 1.3 The XCode toolbar.

On the left-hand side, you'll see the `Groups & Files` list (Figure 1.4).

This list is used to organize the source code, frameworks, libraries, executables, and other types of files in your project.

The list shows several files and groups. Groups can contain other groups and files. You can delete a group as well as create a new one. The group indicated by the blue icon whose name is the same as the name you've chosen as the project name is a *static group*. Underneath it, you see all your headers, implementations, resources (images, audio files, and so on), and other related files. The folderlike

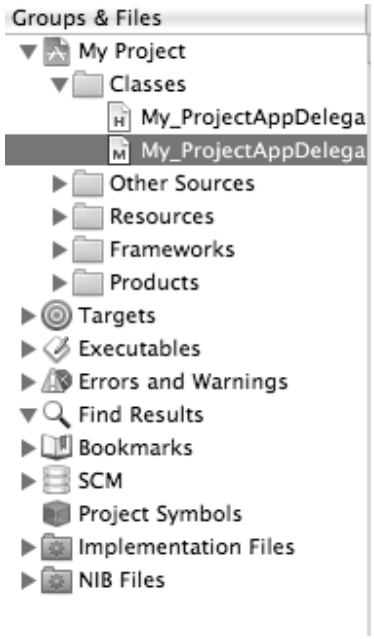

Figure 1.4 The `Groups & Files` list in XCode.

yellow groups act conceptually as containers. You can have containers inside other containers and all files inside these containers live in the same directory on the disk. The hierarchy only helps you organize things. You have full freedom to organize your project's layout as you like. The compiler will pick up the resources, headers, and implementation files when it builds your application.

The other kind of groups that are listed below the project group are called *smart groups*. There are two types of smart groups: (1) built-in smart groups and (2) custom smart groups. The content of the built-in smart groups cannot be customized. Examples of these groups include executables, bookmarks, errors/warnings, and targets. Customized smart groups are shown in purple, and two predefined groups are created for you when you create a new project. The first group is named Implementation Files, and all implementation files are listed underneath it. The other is called Nib Files, underneath which all UI files are listed.

Figure 1.5 shows the `Details` view and the text editor beneath it.

Selecting an item in the `Groups & Files` list will result in its details being shown in the `Details` view. You can go to a full-editor window using `Command-Shift-E`.

1.1.4 Looking closely at the generated code

Expand the `Classes` and `Other Sources` groups. You will notice several files that live underneath these two groups. Click on the `main.m` file and expand to a full-editor view.

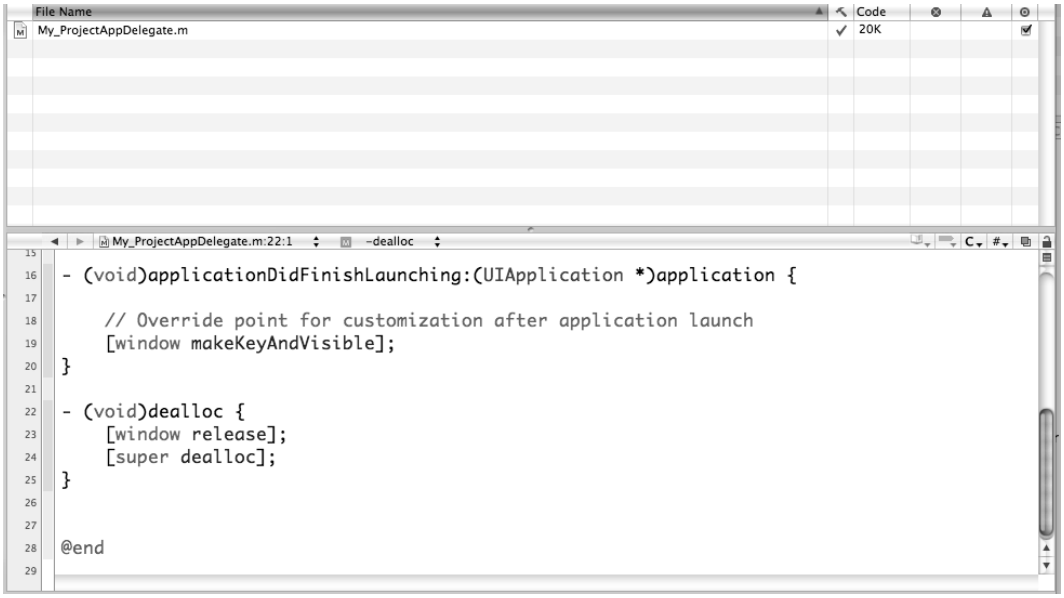

Figure 1.5 The Details view with the text editor view.

The `main.m` file looks very similar to a C file with a `main()` function. As we will see later in this book, all that `main()` does is prepare for memory management and launch the application.

Click on the `My_ProjectAppDelegate.h` file under the `Classes` group. You will notice that the editor changes its content. This file contains the declaration of the application delegate class. Every application that runs on iOS has a delegate object that handles critical phases of its life cycle.

Click on `My_ProjectAppDelegate.m`. This file with the `.m` extension is the counterpart of the previous `.h` file. In it, you see the actual implementation of the application delegate class. Two methods of this class are already implemented for you. The `applicationDidFinishLaunching:` method is one of those methods; it handles a particular phase of the application life cycle. The other method, `dealloc`, is a method where memory used by this object is released. In iOS, you manage the allocation and freeing of memory as there is no garbage collection. Memory management is crucial in iOS development, and mastering it is very important. The first chapters in this book are dedicated to teaching you exactly that — and much more.

The generated files and resources are adequate for starting the application. To launch the application, click on `Build and Go` in the toolbar or press the `Command-Enter` key combination. You'll notice that the application starts in the simulator and it shows only a white screen with the status bar on top. Not very useful, but it works!

1.2 Creating Interfaces

To be useful, an iPhone application needs to utilize the amazing set of UI elements available from the SDK. Our generated iPhone application contains a single UI element: a window.

All iPhone apps have windows (usually one). A *window* is a specialized view that is used to host other views. A *view* is a rectangle piece of real estate on the 320×480 iPhone screen. You can draw in a view, animate a view by flipping it, and receive multitouch events on it. In iPhone development, most of your work goes towards creating views, managing their content, and animating their appearance and disappearance.

Views are arranged into a hierarchy that takes the shape of a tree. A tree has a root element and zero or more child elements. In iOS, the window is the root element and it contains several child views. These child views can in turn contain other child views and so on and so forth.

To generate views and manage their hierarchy, you can use both Interface Builder (IB) and Objective-C code. IB is an application that comes with the SDK that allows you to graphically build your view and save it to a file. This file is then loaded at runtime and the views stored within it come to life on the iPhone screen.

As we mentioned before, you can also use Objective-C code to build the views and manage their hierarchy. Using code is preferred over using IB for the following reasons. First, as a beginner, you need to understand all aspects of the views and their hierarchy. Using a graphical tool, although it simplifies the process, does hide important aspects of the process. Second, in advanced projects, your views' layouts are not static and change depending on the data. Only code will allow you to manage this situation. Finally, IB does not support every UI element all the time. Therefore, you will sometimes need to go in there and generate the views yourself.

The following section teaches you how to use IB. However, for the most part in this book, Objective-C code is used to illustrate the UI concepts. For extensive coverage of Interface Builder, please see Appendix E.

1.2.1 Interface Builder

The project has a basic window resource file. This file can be found under the `Resources` group. Expand the `Resources` group and locate the file `MainWindow.xib`. This file contains the main window of the application. This file is an `.xib` file that stores the serialized objects in the interface. When the project is built, this file is converted to the more optimized format `.nib` and loaded into memory when one or more of the UI components stored in it are requested.

Double-click on the `MainWindow.xib` file to launch IB. IB starts by opening four windows. The first window shows the main window stored in the file. The second window shows the document window listing the different objects stored in the file. The third window is the Library window containing all the UI objects that you can add to the file. The fourth and final window is the Inspector window with its four panes.

The Inspector window shows the attributes of the currently selected object. If you click on an object, the Inspector window shows you its attributes distributed among four different panes. Each pane has several sections. You can change these attributes (such as color, position, and connections) and the changes will propagate to your project's user interface.

The main window of the application is white; let's change it to yellow. Click on the window object in the document window. In the Inspector window, make sure that the leftmost pane is selected. In the View section of this pane, change the background color to yellow as shown in Figure 1.6.

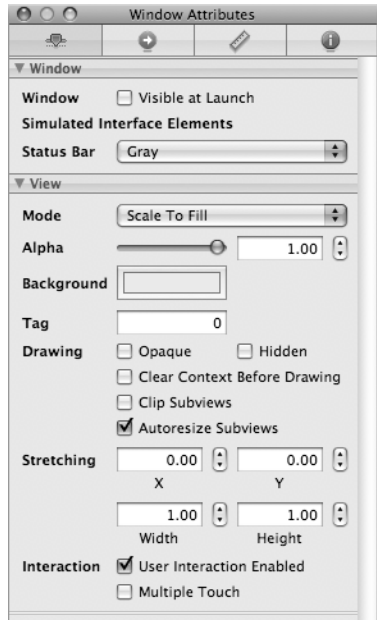

Figure 1.6 The attributes pane in the Inspector window of Interface Builder.

Go to XCode and run the application. Notice how the main window of the application has changed to yellow.

It is important to keep the project open in XCode while working with IB. XCode and IB communicate well when both applications are open.

To build a user interface, you start with a view and add to it subviews of different types. You are encouraged to store separate views in separate `.xib` files. This is important, as referencing one object in a file will result in loading all objects to main memory. Let's go ahead and add a label view to our window. This label will hold the static text `Hello iPhone`.

A label is one of the many UI components available for you. These components are listed under several groups in the `Library`. Select `Tools` → `Library` to show the `Library` window if it's not shown. As shown in Figure 1.7, locate the `Inputs & Values` section.