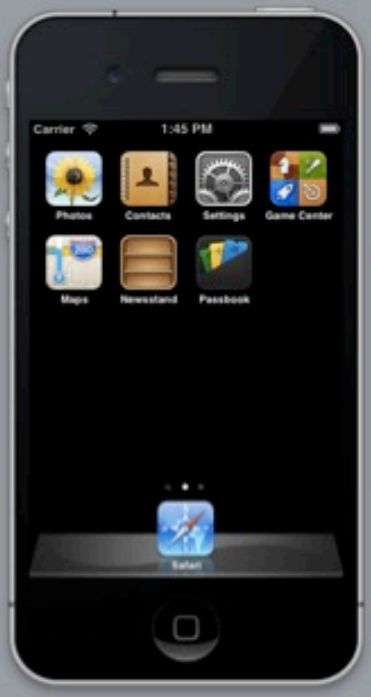
Turn Ideas Into Apps

iPhone/iPad Programming

Learn By Building Real Apps



Asim Jalis

Step-by-Step Guide

iPhone/iPad Programming

Learn By Building Real Apps

Asim Jalis

This book is for sale at http://leanpub.com/iphoneappbook

This version was published on 2013-02-02

Tweet This Book!

Please help Asim Jalis by spreading the word about this book on Twitter!

The suggested hashtag for this book is #iphoneappbook.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#iphoneappbook

To my parents.

Contents

Preface

Why was this book written?

This book evolved out of a course on iPhone/iPad programming[1] that I teach in the Bay Area. The excitement level in the class peaks when we are building apps.

This book draws on the same energy: We are going to build apps. And in the process of making our apps we will learn Objective-C and iPhone/iPad/iOS programming.

This book is dedicated to makers everywhere. People who know the thrill of creating. People who create things and release them.

Who should read this book?

This book targets people who want to learn Objective-C and iPhone/iPad/iOS programming. If you already know another language you will find the content on Objective-C and on the mechanics of designing and building apps useful.

If you have no programming background this book will teach you Objective-C.

In both cases you will find value in this book and in the examples.

How is this book different from other books?

This book is written in a *pull* style. Each new idea is motivated by a question or a problem. The information that follows answers the question and solves the problem. This makes the book more like a conversation or a murder mystery than a lecture. This book will interact with you and make you think.

Who is the author?

Asim Jalis has worked as a software developer at Microsoft, Hewlett-Packard, and Salesforce. He is a software consultant and instructor in San Francisco and has published several iOS apps.

How should this book be used?

This book is highly hands-on and exercise-based. To get the most out of it I recommend firing up Xcode and typing out and running all the examples. Learning happens when a person hand-writes the code, makes mistakes, gets stuck, and then experiences the sweet joy of success. Cutting-and-pasting shortcircuits that process.

I recommend typing out each example and then looking at the book for what to pay attention to and for concepts.

[1] http://metaprose.com

Final words

This book will get you seriously unstuck and on the way to making iPhone apps. If you prefer to not make apps walk away right now.

The mobile platform is amazing and exciting for many reasons. The most compelling one is that you can make apps and demo them wherever you are, to friends, family, and strangers. This book will unleash the app maker inside you. Fasten your seatbelt and get ready for the ride.

Asim Jalis
January 29, 2013

Introduction

Overview

What will this cover?

- Objective-C
- Cocoa
- Xcode

Objective-C

- Apple's programming language for writing iPhone/iPad apps.
- Wood, varnish, and nails for the craftsperson.

Cocoa

- Collection of pre-built components that you glue together using Objective-C to create rich apps without writing a lot of code.
- Pre-manufactured items you buy from Home Depot, like hinges and door knobs.

Xcode

- Toolbox for the app developer.
- Like hammer, chisel, and screwdriver.

Why iPhone/iPad and Objective-C?

- Widely used. Lots of help available online if you get stuck.
- Apple's iPhone/iPad App Store makes selling software easy.
- It's just plain fun to create something out of nothing.

Why Objective-C?

- Great first language.
- Clean and elegant.
- Fast and lean.
- Feels good to make stuff that really works.
- Nice tools.

How is this organized?

- Creating apps is fun.
- Create apps and learn fundamentals along the way.
- Examples are easier to understand than dry abstractions.
- Doing is more fun than listening.

At the end of this:

- You will have created several apps.
- You will have know how to build a broad category of apps.
- You will understand the iOS SDK enough to pick up other APIs.

1 Xcode

In the next sections we are going to create our first app: Dice.

So lets begin by turning on our power tool: Xcode.

1.1 Set Up Xcode

Exercise: Install Xcode

- Go to https://developer.apple.com/xcode/
- Click on *View in Mac App Store*
- Click on *Launch Application*
- In iTunes click on *Install*
- After install completes close iTunes

1.2 Create Xcode Project

Exercise: Create a blank project called *Dice*.

Solution:

- Start *Xcode*
- Click on *Create a new Xcode Project*
- For template choose *iOS –> Application –> Single View Application*
- Click *Next*
- For *Product Name* use *Dice*
- For *Organization Name* use something like *MetaProse*
- For *Company Identifier* use something like *com.metaprose*
- For *Class Prefix* use *XYZ*
- For *Devices* use *iPhone*
- Make sure *only* the *Use Automatic Reference Counting* checkbox is checked
- All other checkboxes should be unchecked
- Click *Next*
- In the file menu click *Create*
- Congratulations, you have created your first Xcode project

1.3 Tour of Xcode

Exercise: View source files, the *XIB* file, the project, and the target settings.

What are the different Xcode panes called?

Name	Description
Xcode Workspace Window	Main Xcode window
Toolbar	Icons along top
Navigator Area	Left pane
Editor Area	Large middle pane
Utilities Area	Right pane
Debug Area	Bottom pane

Which Xcode panes are most used for creating apps?

Name	Description
Project Navigator	For selecting files in folders
Source Editor	For editing *.m* and *.h* files
Interface Builder	For viewing and modifying *.xib* files
Canvas	For designing layout
Document Outline	Thin bar next to Navigator Area

Solution:

- Observe the *Project Navigator* in the left pane
- Click once on each of the *.m* and *.h* files to view them
- Click once on the *.xib* file
- Click once on the project
- Select *Project*
- Select *Target*

What are the different project entities?

Entity	Description
.xib	Interface Builder file describing app user interface
.m	Objective-C source file
.h	Objective-C header file
Target	Settings specific to target device
Project	Settings for the whole project

2 Create Your First App

2.1 App: Dice

Exercise: Create an app for rolling dice.

2.2 Create App UI

Exercise: Create a *UI* or *user interface* for your app.

The UI is the part of your app that appears on the iPhone screen. Think of your app as an iceberg. The UI is the part above the water. The bulk of the app is the Objective-C code below the water.

What are the names of the sub-panes of the Utilities Area?

Name	Description
Inspector Selector	Selects properties to edit
Inspector	Property editor
Library Selector	Selects objects to create
Library	Objects to create by dragging

What do the different inspectors tell us?

Name	What It Says About Object
File	Objects's file
Quick Help	Documentation
Identity	System information
Attributes	Modifiable attributes
Size	Size and constraints
Connections	Connections to other objects

Solution:

- Click on *ViewController.xib*.
- Drag a *Round Rect Button* and a *Label* on to the canvas.
- Resize and place them on the canvas to look beautiful.
- Double-click the button and the label and edit their text.

Object	Text
Button	Roll Dice
Label	-

- Center the label text by selecting the *Attributes Inspector* and modifying *Alignment*.
- Change the font of the label text.
- Select the view–the white box on the left hand of the canvas. Change its background color to white.

- Use the constraint menu on the bottom right to widen the controls.

Exercise: Run the app.

Solution:

- Click *Run*
- Make a change to the UI and click *Run* again

We have the basic UI, however, we haven't wired things up yet, so nothing happens. The app is like a sleep walker. Or a zombie. There's no one home. To fix this we have to add some code.

Notice that you don't need to kill the iPhone simulator to make it pick up the latest changes to the code.

2.3 Model-View-Controller

iOS apps are organized using the *MVC* or *Model-View-Controller* pattern.

The MVC system is made up of three parts.

- The *View* is the UI.
- The *Controller* is the application logic written in Objective-C.
- The *Model* is the source of data: the database or the internet.

The user interacts with the View. This bubbles the message to the Controller. The Controller reads the Model and updates it. Then it responds by making changes to the View.

This cycle repeats until the user terminates the app.

2.4 Connecting View to Controller

Exercise: When the user clicks the button the app should print out the value of the rolled dice.

Solution: We will do do this by wiring up the controller with the UI.

- Enable the Assistant Editor using the Toolbar's editor buttons.
- This will show the header file in the assistant editor window.
- We want to access the label from the code and change its value.
- To do this we connect the label as an *outlet*:
  - Holding down the *control* key drag from the label to the code.
  - For *Connection* choose *Outlet*.
  - For *Name* choose *outputLabel*.
  - Click *Connect*.

- Every time the button is pressed we want the code to be called.
- To enable this we connect the button as an *action*:
  - Holding down the *control* key drag from the button to the code.
  - For *Connection* choose *Action*.
  - For *Name* choose *buttonClicked*.
  - Click *Connect*.
- Look at the header and the source files and see how Xcode automatically inserted properties and the method `buttonClicked`.

2.5 Defining App Logic

Exercise: Implement the code for the controller.

Solution:

- Open *ViewController.m*
- Switch editor to Assistant Editor using Toolbar's editor buttons
- This will make the header file and the source file visible together
- Fill out the `buttonClicked` method so it looks like this

```
- (IBAction)buttonClicked:(id)sender
{
    int diceInt = (arc4random() % 6) + 1;
    NSNumber* dice = [NSNumber numberWithInt: diceInt];
    NSString* face = [dice stringValue];
    [[self outputLabel] setText: face];
}
```

This code fires every time the ViewController receive the `buttonClicked` message,

Here is what the code is doing:

- Generate a random integer between 1 and 6 and save it in `diceInt`.
- Convert `diceInt` from integer to a number and save it in `dice`.
- Convert `dice` from a number to a string and save it in `face`.
- Set the text of `outputLabel` to `face`.

2.6 Logging

Exercise: Add logging to the app.

Logging is a useful for debugging and to make sure the events are flowing correctly in the app.

Solution:

- Add this to the body of `buttonClicked`:

```
NSLog(@"buttonClicked was called");
```