Extracted from:

Test iOS Apps with UI Automation

Bug Hunting Made Easy

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Test iOS Apps with UI Automation

Bug Hunting Made Easy

Jonathan Penn

Edited by Brian P. Hogan

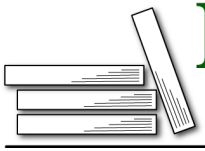
Test iOS Apps with UI Automation

Bug Hunting Made Easy

Jonathan Penn

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Brian P. Hogan (editor)
Potomac Indexing, LLC (indexer)
Candace Cunningham (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-937785-52-9
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—August 2013

Acknowledgments

First, I want to thank my inner circle of authors, who encouraged me to go through the pain to write a book in the first place. Daniel Steinberg, Bill Dudney, Joshua Smith, and Jason Gilmore—thank you. I would have been lost without your example and your terrifying stories.

Thanks to all who submitted feedback and errata throughout the beta process, and specifically those who slogged through the tech reviews and took the time to write up the awesome feedback: Chris Adamson, Heath Borders, Jayme Deffenbaugh, Jason Gilmore, Jeff Holland, Ben Lachman, Kurt Landrus, Kevin Munc, Mark Norgren, Stephen Orr, Julián Romero, Shiney Rossi, Joshua Smith, Daniel Steinberg, Conrad Stoll, Elizabeth Taylor, TJ Usiyan, and Alex Vollmer.

Thanks to CocoaConf for giving me all those opportunities to practice the material in this book—over and over.

Thanks to the team at The Pragmatic Programmers for the resources they provided and for letting me prove myself. Special thanks to my editor, Brian Hogan, for wisely convincing me to scrap the first draft of the book and for fielding my incessant questions.

To my parents, who fed my famished curiosity. To my daughter, Niah, who thinks I work at a coffee shop for a living. To my son, Ian, who thinks I know what I want to do when I grow up. And to my partner, Colleen. She put up with my swinging moods and sleepless nights and surely shed more sweat than I did.

For great justice.

Introduction

We iOS developers have a lot on our minds. We want to build useful and bug-free software for our customers while keeping up with Apple's fast pace. Software development is fraught with trade-offs and, unfortunately, testing our software is often traded away in the crunch time before a release date.

So what's the best way to hunt for bugs in our apps? We spend a lot of our own time manually launching and walking through the features one by one, tapping, swiping…over and over again. This book helps us find a better way.

What Can We Do About It?

Nothing will replace the spot-checking ingenuity of a human tester, but we can certainly automate the important common tasks and free ourselves up to focus on other things. We want to use automated tests to raise confidence while we keep forging ahead and to give us useful information when something goes wrong.

In this book, we're going to focus on testing by scripting interactions through the user interface. This is known as *full stack* or *integration* testing in some circles. We're launching the whole app, tapping and gesturing, waiting for animations, and reacting to results from the screen.

We're going to be strategic with how we apply these tests. Automation testing is a powerful way to smoke out bugs, but it's not without its limitations. These kinds of tests are slow and it's not feasible to test every edge case using this technique. We're not going to cover effective lower-level testing strategies such as unit tests—for more information about that, you'd want to read Graham Lee's book *Test-Driven iOS Development [Lee12]*, or Daniel Steinberg's book *Test Driving iOS Development With Kiwi [Ste12]*. Here, we're looking to test deep slices of the application while answering the question "Did we wire the components correctly?"

We have two ultimate goals with these tests. First, we want to verify correct behavior with *acceptance tests* that list the steps a user would take and the

requirements to consider a feature complete. Second, we want to automate the mundane tasks involved in performance testing. Looking for memory leaks often involves walking through the app and doing the same thing over and over again while recording benchmarks. This is a perfect use case for automation.

Great, So How Do We Get There?

In these pages, we'll be focusing on *UI Automation*, a tool Apple provides that works out of the box and integrates with Xcode. We don't need to install anything to get started and try it out. It was first introduced in iOS 4 as part of Instruments, a robust tool to trace application behavior at runtime. Along with the rest of the instruments available to us, UI Automation gives us a lot of power to assert proper behavior and run extensive performance analysis through different usage scenarios.

Here's where we'll be going:

- Chapter 1, *UI Automation Overview*, on page ?, gets us started by walking through how to capture and play back in the simulator actions we perform on an app. We also take a moment to look at how UI Automation and Instruments work together.
- Chapter 2, *Testing Behavior with UI Automation*, on page ?, builds on the basics and leads you through writing a test that asserts a behavior in the app. We'll take a tour through the automation-scripting interface and learn how we can report failures in our tests.
- Chapter 3, *Building a Test Suite*, on page ?, walks through some simple techniques to start building up a suite of acceptance tests that run one after the other against the application. We'll continue exploring the UI Automation scripting interface and discuss how to group together output from various tests.
- Chapter 4, *Organizing Test Code*, on page ?, explains some good ways to grow our test code in a way that is readable and maintainable. We'll start pulling out reusable pieces into a testing toolbox that we can import anywhere we need it and represent portions of our application screen with special objects.
- Chapter 5, *Maps, Gestures, and Alerts*, on page ?, takes us on a journey underground to learn how UI Automation talks to our application. We'll trigger complex gestures on the map view, alter the way UI Automation sees the elements on the screen, and discuss how best to handle modal alert views.

- Chapter 6, *Strategies for Testing Universal Apps*, on page ?, walks through some scenarios that test the different idioms on iPhone and iPad screens. We'll start a separate suite of iPad acceptance tests while reusing all the testing tools we've built.
- Chapter 7, *Automating Performance Tests*, on page ?, uses the integrated power of UI Automation and Instruments to record benchmarks as the app runs through a variety of performance tests. If you've ever needed to tap, tap, tap over and over again to re-create a memory problem, you'll love this chapter.
- Chapter 8, *Setting Up Application Data*, on page ?, introduces concepts and ideas for bootstrapping the app data in a state that is ready for our tests. We'll discuss good app architectures that make this easier, and look at how environment variables and seed files can inject the information we need into the app at runtime.
- Chapter 9, *Stubbing External Services*, on page ?, helps us deal with the unpredictability of external services. We'll tackle some techniques to fake services at the network layer and even fork our Objective-C code to stub out more-complicated dependencies within the app.
- Chapter 10, *Command-Line Workflow*, on page ?, provides tips to run UI Automation tests from shell scripts. We'll be automating our automated tests, as it were.
- Chapter 11, *Third-Party Tools and Beyond*, on page ?, tours some third-party tools to use with the workflow we discuss in the book. We'll also review useful tools outside of the UI Automation sandbox.

By the end of the book, you'll have a great set of habits you can draw from when you're faced with the unique challenges in your applications.

Follow Along with the Source

Most apps are very complicated state machines with so many possibilities for error that it seems overwhelming. The network, database frameworks, animations, device orientation—all these external and internal dependencies conspire to give us quite a challenge.

We'll face these challenges while studying an actual app throughout the book. By growing a set of test tools based on the needs of a real app, we'll keep ourselves organized and learn to work around the quirks. The source is available for download from the book's website (http://www.pragprog.com/titles/jptios).

Here's the best way to follow along with the code. Each chapter gets its own top-level directory prefixed by the chapter number, like 06-Universal. Each chapter is broken into a series of steps. Every step directory is a complete copy of the app—a snapshot of what the book expects at that point. This is so that you can pick up anywhere in the book and everything will work (or not work if that's what we happen to be exploring). Each snippet of code referenced in this text is annotated to point to the step directory it comes from.

Expectations and Technical Requirements

This isn't a book for iOS beginners. We're going to dive deep into Xcode's build system, the Objective-C runtime, shell scripts, and more. I recommend starting with these books as prerequisite material:

- *iOS SDK Development [AD12]*, by Chris Adamson and Bill Dudney
- *iOS Recipes: Tips and Tricks for Awesome iPhone and iPad Apps [WD11]*, by Paul Warren and Matt Drance
- *Core Data: Apple's API for Persisting Data on Mac OS X [Zar12]*, by Marcus Zarra

I assume you've been through Apple's introductory material, know about how view controllers and memory-management work, and know how to build your own application in the Xcode GUI. We'll be working with at least Xcode 4.6 and iOS 6.1.

Good luck and happy bug-hunting!