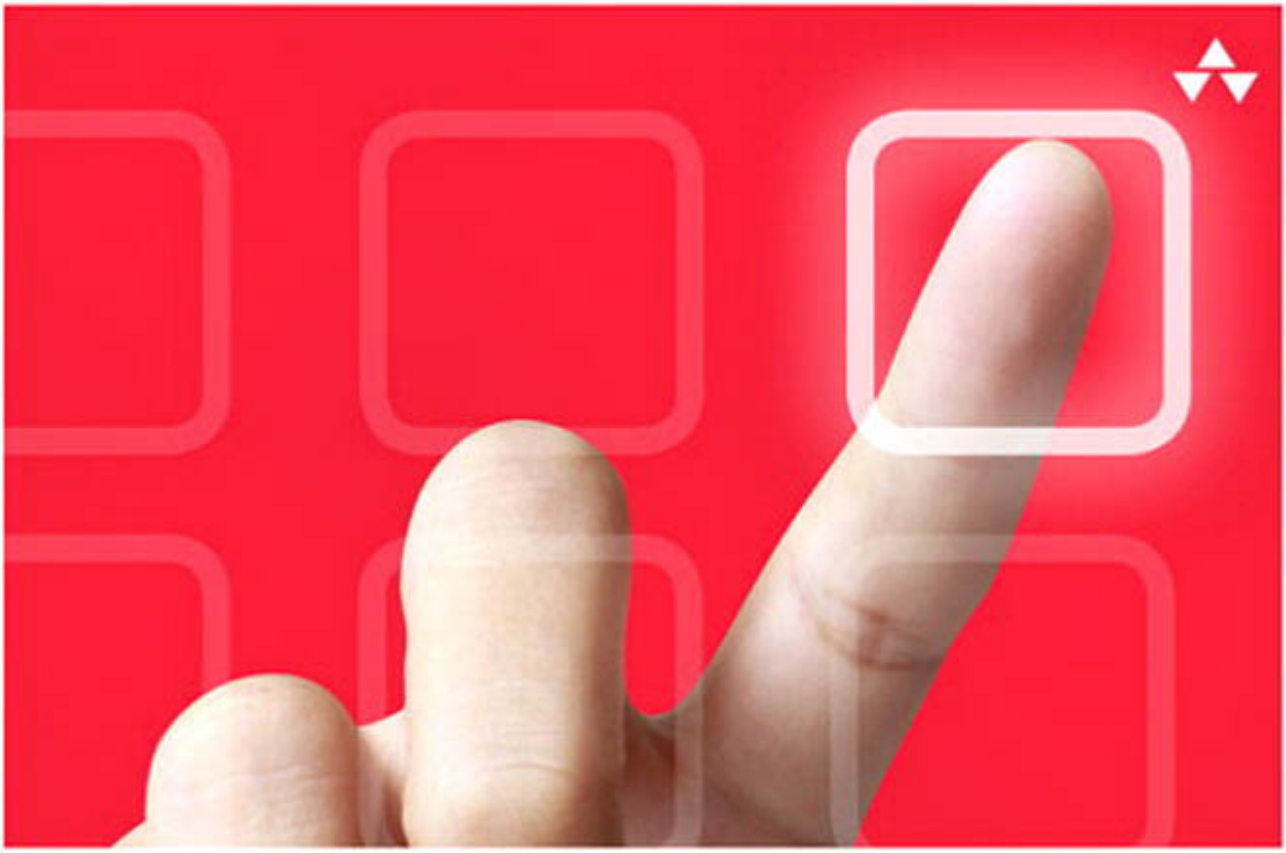


LEARNING iOS DESIGN

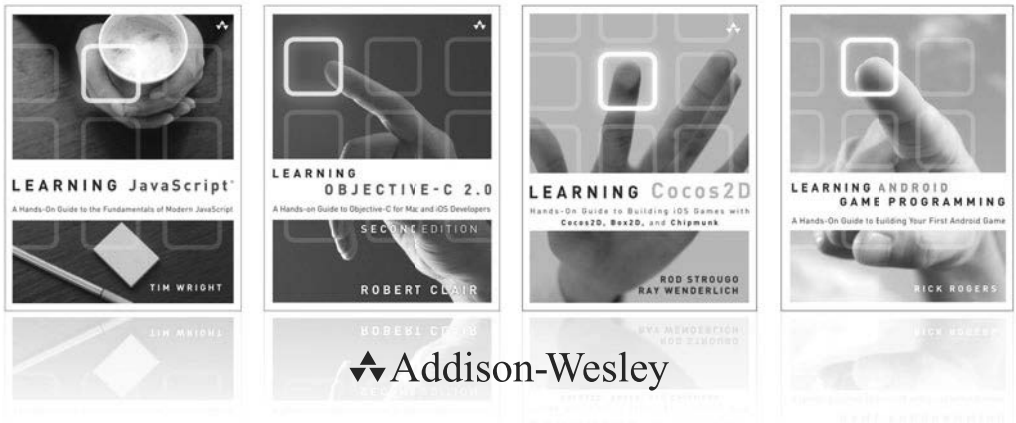
A Hands-On Guide for Programmers and Designers

WILLIAM VAN HECKE

Foreword by LUKAS MATHIS
Author, ignorethecode.net

Learning iOS Design

Addison-Wesley Learning Series



Visit informit.com/learningseries for a complete list of available publications.

The Addison-Wesley Learning Series is a collection of hands-on programming guides that help you quickly learn a new technology or language so you can apply what you've learned right away.

Each title comes with sample code for the application or applications built in the text. This code is fully annotated and can be reused in your own projects with no strings attached. Many chapters end with a series of exercises to encourage you to reexamine what you have just learned, and to tweak or adjust the code as a way of learning.

Titles in this series take a simple approach: they get you going right away and leave you with the ability to walk off and build your own application and apply the language or technology to whatever you are working on.

Learning iOS Design

A Hands-On Guide for Programmers and Designers

William Van Hecke

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

Van Hecke, William.
Learning iOS design : a hands-on guide for programmers and designers / William Van Hecke.
pages cm
Includes index.
ISBN-13: 978-0-321-88749-8 (pbk. : alk. paper)
ISBN-10: 0-321-88749-2 (pbk. : alk.paper)
1. iOS (Electronic resource) 2. Application software—Development. 3. iPad (Computer)—Programming. 4. iPhone (Smartphone)—Programming. I. Title.
QA76.774.I67V36 2013
004.167—dc23

2013010043

ISBN-13: 978-0-321-88749-8
ISBN-10: 0-321-88749-2
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, June 2013

Editor-in-Chief
Mark L. Taub

Senior Acquisitions Editor
Trina MacDonald

Development Editor
Sheri Cain

Managing Editor
John Fuller

Full-Service Production Manager
Julie B. Nahil

Project Editor
Anna Popick

Copy Editor
Betsy Hardinger

Indexer
Jack Lewis

Proofreader
Anna Popick

Technical Reviewers
Jon Bell
Jim Correia
Lukas Mathis

Editorial Assistant
Olivia Basegio

Cover Designer
Chuti Prasertsith

Compositor
Rob Mauhar

❖

To Buzz and CeeCee; Touichi and Risako

❖

This page intentionally left blank

Contents at a Glance

This page intentionally left blank

Contents

This page intentionally left blank

Foreword

When Apple introduced Mac OS X, Mac users' feelings were ambivalent. Sure, this looked like a fantastic operating system, but a huge part of what made the Mac unique was its software. Photoshop, Illustrator, Claris Works, MacPaint—these were the reasons we used Macs. And with Mac OS X, all of these applications effectively stopped working. There were few native applications for Mac OS X, and fewer still that weren't horrible.

There was, however, one company that consistently developed fantastic software for Mac OS X right from the start. And they kept doing it. For the last decade, The Omni Group has been a sure bet for quality products. Applications like OmniGraffle combine ease of use and sheer power in a way that is unique, yet feels completely natural. On the one hand, these applications are incredibly accessible. It takes very little to create fantastic output. On the other hand, they have great depth. Recently, the Omni Group has expanded their reach to iOS, and they've done something almost nobody else outside of Apple has achieved: they've brought their applications to the iPad in a way that makes them feel native to these portable touchscreen devices, but doesn't diminish their power and depth.

I'm probably not the only designer who has more than once looked at applications like OmniOutliner, OmniGraffle, or the somewhat exorbitantly named OmniGraphSketcher and wondered to themselves: How do they do it? How do these people consistently create software that seems to effortlessly present incredibly powerful features in a way that is easily accessible, and a pleasure to use? And even more puzzling, how do they manage to achieve this feat on iOS, a platform famous for its abundance of shallow, poorly designed, one-trick-pony, cash-grab apps?

Well, today's your lucky day, because you're holding the answer to this question in your hands. My friend Bill, who wrote this book, happens to be Omni's User Experience Lead. And he's lifting his kilt, just for you.

I first consciously heard of Bill when he became Internet-famous for talking about Omni's 1:1 replicas of iPads made from wood, cardboard, Plexiglas, and 3-D-printed parts. Who would want to make 1:1 replicas of iPads? Well, Apple had announced the iPad, but had not yet started shipping it. Having already started designing apps for the iPad, Bill's team needed to get an idea for how these apps would feel on an actual device. At this point, less dedicated people would just postpone the whole thing for a few months. But not Bill's team. They went ahead and made their own iPads.

Most UX designers eventually manage to come up with a design that works well. It's this kind of relentless dedication to detail, this kind of work ethic, though, that is

the difference between a designer who can come up with a good design, and one who will come up with a mind-blowingly awesome design.

But there's something else that makes Bill unique among his peers. Any designer will tell you that their goal is to make the product they're working on beautiful and easy to use and efficient and pleasant. But Bill goes one step further. His goal isn't just to make apps user-friendly, but to touch the user's soul, to help people make more beautiful things, be more successful, and be happier. In one of his presentations, he recounts how one man converted his classic VW Beetle into an electric car with the help of OmniGraffle. To Bill, that's the ultimate goal. Software design isn't just about making an application easy to use, it's about making the application have a positive impact on people's lives. It's about helping people be better.

This book contains everything you need to know to create awesome, life-altering applications, just like Omni's. While it's targeted at iOS designers, you're going to learn a lot from reading this book regardless of the platform you design for. I pride myself on knowing a lot about design, but when reading this book, I probably didn't encounter a single page that didn't offer at least one interesting idea, new concept, or clever design technique. From learning how to make your application more forgiving to a section on how pricing influences how people perceive your app (yep, its price is part of the app's design), you're in for a treat.

Even better, this book doesn't just offer invaluable content that will forever change the way you design applications, it's also written in a way that prevents you from putting it down. So grab a hot cup of cocoa, put on your favorite music, and settle down into your most comfortable chair, my friend, because you'll be sitting here, staring at this book, for quite a while.

Enjoy it.

—Lukas Mathis, ignorethecode.net; author of *Designed for Use: Create Usable Interfaces for Applications and the Web* (Pragmatic Bookshelf, 2011)
March 2013

Preface

Hello

It took a while for the world to notice, but design really matters.

A perfect story of the power of design can be found by traveling back to April 2007 to eavesdrop on a chat with Microsoft CEO Steve Ballmer. Apple's Steve Jobs had announced the iPhone that January, and everyone had had a good while to process the announcement and decide what they thought of it. Ballmer, in an interview with *USA Today,* opined on the iPhone's chances to make a dent in the well-populated smartphone market: "There's no chance that the iPhone is going to get any significant market share. No chance."

I'm not normally one to indulge in schadenfreude, but the wrongness of that prediction is too illuminating to ignore. iPhone went on to become an icon that redefined the public's concept of what a mobile phone is, and nearly every "smartphone" on the market takes inspiration from it. Its sibling, iPad, finally popularized the stagnant tablet concept and is on its way to replacing the traditional desktop or notebook computer for millions. iPhone and iPad each own about half of the market share of their respective markets. The App Store model has redefined the way people buy software and has paid out more than $7 billion to third-party developers. As of the beginning of 2013, nearly half a billion iOS devices have been sold.

Why? How did iOS become so successful? What did Ballmer and the rest of the early-2007 iPhone scoffers miss? Ask any authority who followed the story closely to pick one word to describe Apple's advantage, and they'll say *design*. (Some cynics might say *marketing,* but they're wrong.)

iOS is arguably the first technology platform to truly put design first. Instead of the puffed-up and bulleted feature lists, the contortions to accommodate legacy systems, the assumptions about how a phone was supposed to look or behave, and the obsession with being the first to the market, iPhone prioritized beauty, responsiveness, and fun. (And anything that Apple couldn't get just right was omitted until they could.) This view of design is about creating happiness, about cultivating a relationship with the user, about imagining the most positive *user experience* possible and then doing whatever it takes to produce that imagined outcome.

You could almost say that iPhone refused to compromise on its user experience. But as this book argues, all designs are compromises. Surely, countless tradeoffs and tough decisions were made in the process of bringing iOS into being. But what's important is

that wherever possible, those compromises erred on the side of paying attention to detail, abandoning conventional wisdom, and putting in more work to make users happier.

Not solely because of Apple and iOS, but in large part, the world is learning that design counts. It's getting harder to compete without good design. It's harder to find good designers than it is to find good engineers (and that itself is pretty hard). Well-designed software really can improve people's lives, help them be more productive, and yes, make them happy. This book aims to give you the practices, examples, and advice you need to make it happen yourself.

You're a Designer

Design is deciding how a thing should be. In every act of design, that decision-making is done to accommodate constraints and to satisfy the needs of some audience or "user." The needs are paramount, because an artifact that doesn't do anything useful for anyone is more a piece of art than a design. And the constraints are your friends, because they narrow the space of possibilities, making your job much more approachable. Almost everything you think about and do as a designer can be narrowed down to these concepts: How are you serving the needs of the user? How are you working within the constraints?

Everything artificial was designed by someone. Most of the time you don't think about the people who decided how the things around us should be: the height of a chair's seat, the shape of a battery charger, the hem of a blanket. That blissful ignorance is the goal of many designers. If people don't think about the design of an object, the designer has probably done a fantastic job. More than two thousand years ago, Ovid said it like this: *Si latet, ars prodest*. If the art is concealed, it succeeds. That's one to print and hang on your wall.

If you've ever made something, then you're a designer. Ever built a couch fort? Arranged some flowers in a vase? Sketched a map for someone? Whether or not you thought very much about it, whether or not you followed well-researched principles, you designed that thing. That's design, with a lowercase *d*. You could take that approach to designing an iOS app, but the result isn't likely to be compelling. Books like this one aim to help you do Design with a capital *D*. That means absorbing and imagining as much as you can about how things could be better. It means making the smartest, most informed decisions possible about the needs and constraints involved. And it almost always means creating plans, sketches, and models along the way to a final product. The good news is that you can get there from here, one step at a time, always experimenting and learning as you go.

Meet the Book

This book introduces and explores the topic of designing iOS apps, even if you don't consider yourself a designer (yet). Even if you've never taken an art or design course, if you consider yourself to have more of an engineering or analytical mind than a

creative one, or if you're mystified by what actually goes on in the process of design, you're very welcome here.

At conferences, I've presented the topic of design to a largely engineering-minded audience. Lots of programmers know that they should care about design, but the practice of design seems from the outside to be mysterious or even arbitrary, leaving them disillusioned or apathetic about it. But after some demystification and conversation, some folks have told me that they finally get why design is important and how they can think about it systematically.

This book presents the art and science of design in an accessible, sensible way.

Part I: Turning Ideas into Software steps through the phases of design, turning a vague idea for an app into a fully fleshed-out design. It goes from outlines to sketches to wireframes to mockups and prototypes. Each step of the way, you'll find advice about how to think carefully, critically, and cleverly about your project. Each chapter concludes with exercises conceived to encourage you in planning the design of your own app. Part I includes the following seven chapters.

- **Chapter 1: The Outlines**—This is all about planning, writing things down, and making sense of your app idea. You'll learn about the ways you can use structured thinking and writing to figure out what your app is about and stay on track throughout the project.
- **Chapter 2: The Sketches**—Sketching is the central activity of design. It's all about getting ideas out there and seeing where they lead. You can never know the merits of an idea until it's on a page, a whiteboard, or a screen. This chapter will help you sketch with the right blend of adventurousness and discipline.
- **Chapter 3: Getting Familiar with iOS**—Understanding the constraints of the platform is crucial. iOS offers a versatile kit for building interfaces and experiences; you should know it well enough to decide when to take advantage of it and when to diverge from it.
- **Chapter 4: The Wireframes**—Eventually you need to turn your sketches into precise, screen-by-screen definitions of how the app should be organized. A wireframe is a document that specifies layout and navigation without getting bogged down in pixel-perfect styling just yet.
- **Chapter 5: The Mockups**—It's not the only concern of design by far, but it matters what your application looks like on the surface. In this chapter you'll break out the graphics apps and learn how to assemble beautiful assets into a convincing, pleasant whole.
- **Chapter 6: The Prototypes**—Sometimes a static drawing of an interface is not enough. You need to know how it behaves. This chapter is all about simulating and testing the interactions that make up your app.
- **Chapter 7: Going Cross-Platform**—Plenty of apps exist not as completely standalone experiences, but as parts of a multiplatform suite. This chapter explores the concerns you'll need to deal with if you want to build the same app

for more than one device. It uses an app that appears on iPhone, iPad, and Mac as a case study to illustrate how a single idea can wear three different interfaces.

Part II: Principles presents universal principles that apply to any design and that you should follow if you want to craft an effective app that people will appreciate and even love. To make sure your app works on every level, each chapter in this part is based on one of the three levels of cognition identified by psychologist Donald Norman. Many of these principles are applicable to all software design, but here they're tailored to the strengths and challenges of iOS. The exercises for each chapter present sample situations to help you learn how to apply each principle.

- **Chapter 8: The Graceful Interface**—This chapter examines the visceral level of cognition, which relates to the way people feel from instant to instant as they interact with software. It deals with things like touch input, timing, and feel. Most of the concerns here are subconscious. Users may not notice them, but they subtly affect how pleasant the software is to use.
- **Chapter 9: The Gracious Interface**—Here you'll learn about concerns at the behavioral level of cognition. That means how users make decisions moment to moment and how the app communicates possibilities and status. The chapter also discusses how the app can encourage a sense of adventure so that users feel welcome and safe as they explore its possibilities.
- **Chapter 10: The Whole Experience**—The biggest, vaguest, most intangible, and most important level of cognition is the reflective level. This chapter explains how people feel about your app in the long run: whether they rate it well, whether they recommend it to friends, whether they respect you as a developer, and whether they'd buy from you again. Happiness is the ultimate goal.

Part III: Finding Equilibrium is meant to function as a reference, inspiration, and exploratory guide to the various decision points you may encounter in designing an app. It embraces the concept that all designs are compromises and that many decisions have no single correct answer. This means that many answers to the same design problem can coexist, and every design, no matter how unfashionable or unsophisticated it seems, has something to teach (a fact that many critics seem to forget). You can look at each chapter's opposed approaches as a sort of slider control, with a continuum of answers between the extremes at either end. For each challenge, a smart designer like you should seek an answer that works best for your app's unique philosophy. Over time you may find yourself preferring one side of a given slider over the other. Maybe you like to err on the side of focused rather than versatile. Or perhaps you'd rather seek the Aristotelian golden mean, straight down the middle. That's great. That's what it means to have a style. Each type of decision is illustrated by examples of different solutions to the same problem, depending on the angle you prefer. The exercises encourage you to find your own favorite solution for a situation that may have several possible answers.

- **Chapter 11: Focused and Versatile**—One of the biggest decisions you need to make about your app is its scope. Do you want to do one thing flawlessly, or many things competently? What's feasible depends on the resources available and your ability to be aggressive about defining what you expect of the project.
- **Chapter 12: Quiet and Forthcoming**—When most people talk about a design being "simple," what they usually mean is that it's in good order and presents an understandable amount of information and control at once. In contrast, designs feel empowering when they simultaneously present as much as possible. This chapter describes how to control the apparent simplicity of your app from screen to screen, depending on the emotion you prefer to evoke.
- **Chapter 13: Friction and Guidance**—Part of the job of a software designer is to make many things possible, but also to gently guide people through an experience. This chapter is about the ways an interface puts down grooves that encourage a user to move this way or that way next, or slow down before taking the next step.
- **Chapter 14: Consistency and Specialization**—Differentiating yourself from the rest of the apps out there is both an advantage and a risk. When you think of well-designed apps, the examples that come readily to mind are the ones that break from convention and get away with it. But respecting the established guidelines is usually the wiser path. This chapter will help you decide when to stick to the script and when to diverge.
- **Chapter 15: Rich and Plain**—The visual styling of an app is the most conspicuous outward manifestation of its design. Independent of its functionality, your app can look extravagant or subdued, lifelike or digital. This chapter will help you tune the depth, color, and realism of your interface to set its tone and personality.

Meet the Web Site

The web site for this book is **http://learningiosdesign.com**. There, you will find resources such as the Photoshop and OmniGraffle source files for the examples given throughout the book. You can also offer feedback about the book and find updates of its content.

You and Your Team

You can follow this book as you work on your own app idea, especially by working through the practices described in Part I. Even if you don't yet have an app project, or if your app already exists and you want to revise it for a new version, you should be able to benefit from the book. Parts II and III are compatible with dipping into for inspiration or advice.

From time to time, the book may talk as if you are a designer working with a software engineer or a team of engineers. That of course doesn't need to be the case. Maybe you're one of that noble species, the lone programmer/designer hybrid. Maybe you're a product manager looking to understand design better. It doesn't really matter; whenever this book mentions "your engineers," it's fine if that means you!

Art/Science Duality

Design is full of what are called "wicked problems": they're difficult to define, they're impossible to come up with definitive answers to, and they're never finished. That's likely to spook some people, but it's also what makes design so much *fun*. You never know what you're going to get. There's always some way to improve on your work. Everything is a matter of taste, and yet some answers are unequivocally better than others. There's no recipe, and yet there are morsels of wisdom and inspiration to be found everywhere.

Design is an art. And it's a science. And it's neither. Steve Jobs liked to say that what Apple does falls "at the intersection of technology and liberal arts." You may find your team arguing about how to make a decision. One side is showing numbers; usability test metrics clearly indicate that design *A* is more efficient than design *B*. The other side is arguing that based on aesthetics, design *A* just doesn't *feel* right. Who wins? Maybe it's one of those two options; maybe it's a third, new option. Figuring it out is part of the thrill of design.

You could take a completely scientific approach, refusing to budge on anything until you've run a statistically significant study. You could also take a completely artistic approach, following your muse and composing your personal magnum opus in app form. But you won't get very far with either one alone—data and heart both matter.

Inspiration Is Everywhere

This book can give you specific advice on specific topics and situations that occur often in the work of designing apps for iOS. But your growth as a designer depends, more than anything else, on your willingness to absorb inspiration from around you. Pay attention to all kinds of design: graphics, interiors, architecture, games, anything. Read widely: psychology, art, history, biology, everything. The most seemingly irrelevant knowledge may end up informing your work as a designer someday, in some oblique way. If you do nothing else, use lots of well-regarded apps and think about what makes them successful. The more you examine and ponder great work of all kinds, the better you'll get at it yourself.

Again, growing as a designer is a lifelong journey, but here is a necessarily short list of reading material to get you started. Some of these books are mentioned again in the chapters where they're especially relevant.

- *Universal Principles of Design* by William Lidwell, Kritina Holden, and Jill Butler—A delightful collection of 125 concepts that apply to all categories of design. Very compatible with flipping through for quick inspiration.
- *The Elements of Typographic Style* by Robert Bringhurst—One of the most carefully built, wisdom-packed books of all time. Yes, Bringhurst will make you knowledgeable about type, but he will also inspire you with his methodical, tasteful approach to design in general.
- *Visual Explanations: Images and Quantities, Evidence and Narrative* by Edward Tufte—Or any of his four main books, really. Tufte tends to lean toward information design for print, but the principles he espouses should be useful to anyone who has any interest in making things understandable and beautiful.
- *Designing Interactions* by Bill Moggridge—This book is a collection of captivating interviews (included on DVD) from original Macintosh software lead Bill Atkinson to legendary game designer Will Wright.
- *Sketching User Experiences: Getting the Design Right and the Right Design* by Bill Buxton—Much of the reverence that technology designers have for the practice of sketching can be credited to Buxton. Sketching is good for your brain and good for your work.
- *The Design of Everyday Things* by Donald Norman—A classic that has stood the test of time. This book pioneered the dissatisfaction with poorly designed experiences and set the stage for a generation of designers to make the world a more agreeable place to live in.
- *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests* by Jeffrey Rubin and Dana Chisnell—If you're interested in the scientific side of design, this is an excellent walkthrough of the procedures and principles of collecting data from a sample of the target audience using your app.
- "The Nature of Design Practice and Implications for Interaction Design Research" by Erik Stolterman—A brief academic paper, chock full of references to other influential papers, about what design really is and how to deal with its complexity.
- *Basic Visual Concepts and Principles: For Artists, Architects and Designers* by Charles Wallschlaeger and Cynthia Busic-Snyder—A solid grounding in perception and the construction of visuals.
- *Revolution in the Valley: The Insanely Great Story of How the Mac Was Made* by Andy Hertzfeld—This book is a treasure trove of firsthand anecdotes about the culture and creativity surrounding the development of the original Macintosh. If it doesn't get you excited about making technology, nothing will.
- *How the Mind Works* by Steven Pinker—A comprehensive tour of what we understand so far about human psychology. Not directly related to software design, but a surprising source of insight into how people think and why design principles work the way they do.

- *Thinking, Fast and Slow* by Daniel Kahneman—An up-to-date psychology book about how people pay attention, judge situations, and make decisions. Another surprisingly enlightening read for science-minded designers.

And here are a couple of things that aren't books.

- "Inventing on Principle"—A one-hour talk by Bret Victor, interaction designer for iPad (among many other impressive accomplishments). Victor has among the most thoughtful and inspirational minds in technology design, and this talk is a fantastic place to start learning from him. This is the sort of talk you'll want to come back to once a year or so.
- Ideo Method Cards—A deck of cards from the legendary product design firm Ideo. Each card describes a "user-centered" practice that can be of use to designers working through an interesting problem. You can casually flip through the deck for ideas, assemble a mini-deck for a given project, or make up your own ways of getting the most out of them.
- Oblique Strategies—A set of cards, each bearing an enigmatic phrase meant to motivate and give direction to a person facing a creative problem. They were originally created by Brian Eno and Peter Schmidt for musicians, but creative people of all kinds have since found them useful for breaking through difficulty. The cards themselves are rare, but plenty of web- and app-based editions are available.

I found these resources helpful. Hopefully some of them will be at home in your own garden of influences and inspirations.

Now…let's make some software.

Acknowledgments

Turns out writing a book is hard! Mountains of thanks go out to all these people for making it possible.

Thanks to Barbara Gavin and Erica Sadun for taking a chance on a shy and inexperienced speaker and inviting me to speak at the Voices That Matter series of conferences, which eventually led to this book project. Thanks to Trina MacDonald at Addison-Wesley for guiding me through the writing process. Thanks to Betsy Hardinger for editing that makes me seem like a much better writer than I am. Monumental thanks to my review board: Lukas Mathis, Jim Correia, and Jon Bell; my trust in their wisdom is the reason I've been able to maintain confidence in this endeavor.

Thanks to all my colleagues at the Omni Group for giving me the chance to make good software and talk to brilliant people all day *as my job*. Every day, I feel as if I'm getting away with something. Thanks to my instructors and classmates at the University of Washington's Human-Centered Design & Engineering professional M.S. program, where I've finally been able to get an academic grounding in the thing I've been doing all this time. Thanks to my dear friends in #rosa for their endless support and encouragement.

Admiration and thanks go to Yasunori Mitsuda, whose Xenogears albums provided the soundtrack that kept me pushing keys. Thanks, too, to the various coffee shops of Seattle, for providing the perfect writing environment.

It seems as if every book's acknowledgments page mentions family members' patience; now I understand why. Copious gratitude and love to my wife, Hiroko, for her steadfast patience and support. Ultimately, everything is thanks to her.

This page intentionally left blank

About the Author

Since 2004, **William Van Hecke** has been User Experience Lead at the Omni Group, one of the world's most accomplished and affable Mac and iOS developers. Bill got his start designing software by reverse-engineering his older brother's text adventures in MS Basic on the Macintosh Plus, and then graduated to creating HyperCard games to mail to his cousins on floppy disk.

Bill's primary hobby is hobby-collecting: reading fiction and science; playing bass guitar; appreciating, translating, and developing niche video games; studying the Japanese language; mastering tabletop gaming; and exploring 3-D modeling. You can find Bill on Twitter, prattling on about these topics and more (@fet).

This page intentionally left blank

We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write me directly to let us know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone or email address.

Email: trina.macdonald@pearson.com

Mail: Reader Feedback
Addison-Wesley Learning Series
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our web site and register this book at **informit.com/register** for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

3

Getting Familiar with iOS

A major part of getting good at designing for iOS is simply coming to understand what the platform has to offer. Much of your design work will consist of choosing from a set of standard navigation schemes and controls. Occasionally you'll need to dream up a custom element, but even your original designs should be in the spirit of the platform.

Although you may see these standard elements and behaviors in the apps you already use every day, you might not realize the reasons Apple and savvy third-party developers use them the way they do. This chapter supplements the *iOS Human Interface Guidelines* with further explanation and advice about the standard choices available. Then I talk about how to create your own custom designs that feel at home on iOS.

After you read this chapter, you'll be ready to start wireframing. While outlining and sketching, you can get away with scrawling a vague blob in the general vicinity of a feature and saying something like, "We'll need to provide some way to turn this setting on and off." When it comes time to wireframe, though, you'd better know your options for a setting that needs a two-way toggle (probably a switch, a segmented control, or a table with checkmarks, depending on the needs of the design).

Navigation: Screen to Screen

Put too simply, the primary challenge of wireframing is figuring out how to fit a list of features onto a series of two-dimensional screens. Part of that challenge is providing navigation between the screens in a way that makes sense and is easy for users. Let's look at some dependable ways to construct a sensible navigation scheme for your app.

Navigation Controller

A **navigation controller** is the most common way to get between screens on iOS (see Figure 3.1 for an example). A **navigation bar** at the top of a screen indicates the current location and contains a back button; rightward-pointing chevrons in the content area offer ways to proceed down the hierarchy. This arrangement allows for any number of branching paths, with a consistent way of getting back up to the top.

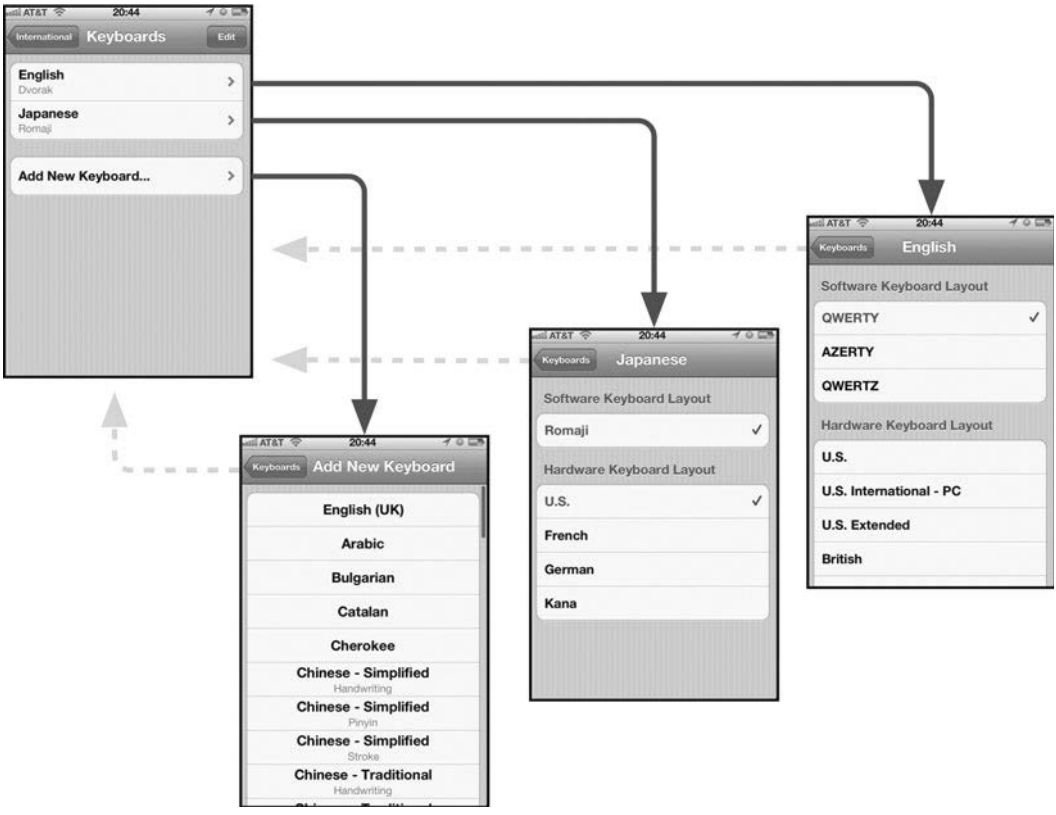


Figure 3.1 The Keyboards branch of the navigation controller hierarchy in the iPhone Settings app. Tapping a table cell with a chevron takes you to a new screen. Tapping the back button in the navigation bar takes you back a screen.

The result is a navigation scheme in which a user can scroll vertically on tall screens of information, and move horizontally to step through the greater hierarchy of screens. Navigation controllers work well only for a limited number of levels. If your hierarchy requires users to routinely delve four or five levels deep, you may need to flatten it out. (Chapter 12, Quiet and Forthcoming, explains how to flatten interface hierarchies.)

Navigation controllers are very familiar to iOS users. They trace their heritage all the way back to 2001, on the original iPod, where selecting a row required spinning a physical wheel. They're still the most reliable, most predictable way on iOS to present a treelike hierarchy of information. Users take it for granted that many iOS apps have a navigation bar at the top of each screen, and that's the first place they look to check which screen they're on and how to go back a screen. Yes, the navigation controller is the go-to workhorse for getting around in an iOS app, and it'll seldom do you wrong.

Most navigation controllers use standard table views (described later in this chapter) to list the options on each screen, but that's not the only approach you can try. (See Figure 3.2 for examples of other approaches.) Following are the only strong rules for creating a navigation controller experience.

- Delving down a level should involve tapping an element that either bears a rightward-pointing chevron or otherwise presents an obviously tappable piece of content that the user would want to navigate to.
- There should be a recognizable back button in the upper left, labeled with the name of the screen that it takes you back to (not with the word "Back").
- Moving between screens should use a horizontal slide animation—the content slides leftward for delving down, and rightward for going back up.

As long as you follow those guidelines, the navigation experience should feel comfortable and familiar to users. In that spirit, here are a few ways of presenting options that differ from the ordinary table view.

- A map, where users can tap a pin to open a label and then tap a disclosure button (bearing a chevron) to navigate to a detail screen. The Maps app on iPhone is an obvious example, because a map is a much more appropriate way to display geographical data than a list would be.
- A collection of big, expressive images, especially when there is a strong case for relying on images rather than text to identify your items. This would be an exception to the need for rightward chevrons, because they would unbalance the composition of each icon. Podcasts' cover view is an example; people react more immediately and emotionally to colorful cover images than to a list of titles.
- A grid of carefully laid-out previews of content. For an app that focuses on a specific kind of content, this approach can be far more expressive and inviting

Figure 3.2 Navigation controllers don't need to use ordinary table views, as shown on these Maps, Clock, and Podcast screens.

than a simple list of titles. Instapaper offers cleanly arranged previews of web articles to help you decide which one to read.

Split View

Only available on iPad, a **split view** offers a way to present navigation and content at the same time, something that helps flatten the navigation hierarchy. Most iPad apps that have a branching hierarchy work well with a split view. The relationship between the two sides is simple: what's selected on the left pane appears in detail on the right pane. Either side can have a navigation controller with ordinary horizontal navigation, but both shouldn't have navigation. (Really, they shouldn't. It has been tried, and the result is a confusing mess.) This means that you have two options for spreading navigation across a split view (see Figure 3.3 for an illustration of these two approaches).

- One is like the Settings app, where the sidebar always portrays the top level, and the content area can navigate. This makes it easier to jump around between lots of top-level items and then delve into detail for items on the right side if necessary.
- The other option is like Mail, where the sidebar can navigate, and the content area always displays details about what's selected on the left. This is a good approach when you have a well-defined, consistent type of content being displayed on the right, such as emails. People can navigate around their mailbox structure, and as soon as they tap an email, the right side updates to show it.

Remember that when you use a split view, you need to decide what happens in portrait orientation: either the split should remain visible, or the sidebar should be

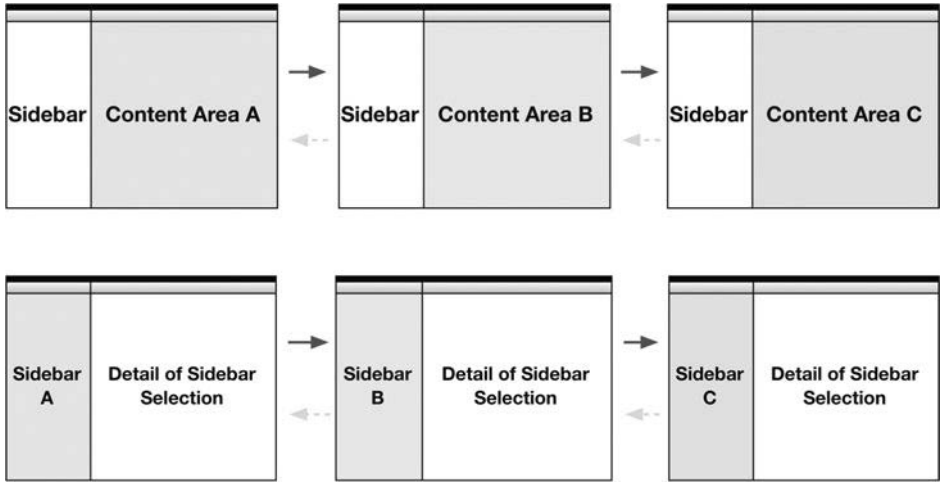


Figure 3.3 At top, Settings-like split view navigation. At bottom, Mail-like split view navigation.

hidden, sliding in when summoned by the button in the upper-left corner. The answer is easy when you use Settings-style navigation as just described: keep the sidebar visible. The back button in the upper-left means you wouldn't have a place to put a sidebar toggle button. (Some apps, such as Facebook, do it anyway. The sidebar is hidden, and it's unavailable at any level other than the top.)

For a Mail-style app, the answer depends on how important it is to keep the sidebar visible at all times versus how likely it is that the user will want to focus on the content area. In Mail, the sidebar is hidden in portrait orientation because users want to focus on a generously sized message area. (See Figure 3.4 for an illustration of these options.)

Tabs

A **tab bar** provides ever-present top-level navigation at the bottom of the screen. It's perfect for an app that needs to provide quick access to a few distinct top-level screens. The quintessential example is the Music app on iPhone: it offers tabs for Artists, Playlists, Audiobooks, and so on. Listeners on the go probably want to quickly jump to a certain category and then quickly navigate to the content they're interested in. The tab bar means that even if they last left the app several levels deep in the Artists hierarchy,

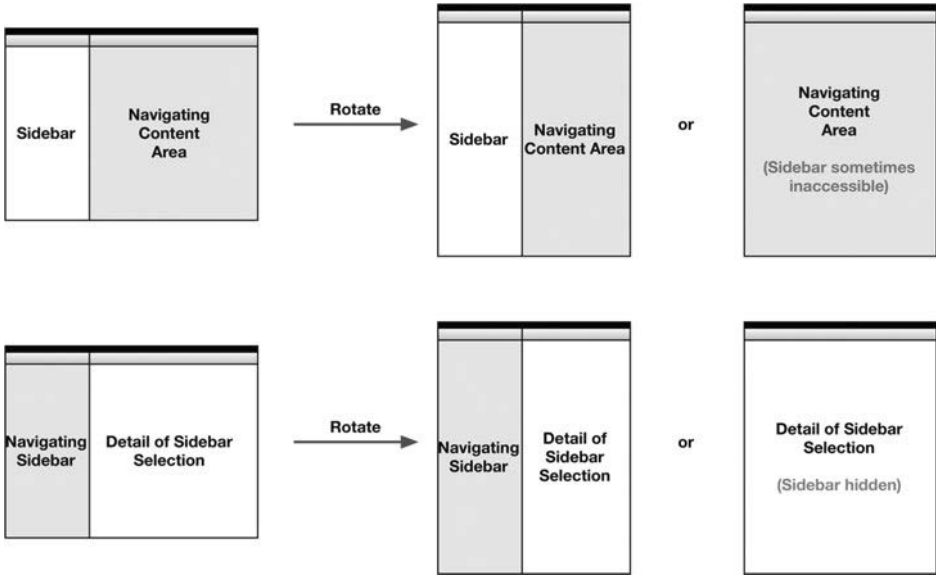


Figure 3.4 Portrait orientation and split views. At top, a navigation controller in the main content area. Hiding the sidebar means the user must navigate all the way to the top level to get the sidebar back; keeping the sidebar is the kinder choice. At bottom, a navigation controller in the sidebar. Hiding the sidebar helps the user focus on the content area; keeping it gives the user easier access to navigation.

they can still immediately tap the Audiobooks tab and start navigating that hierarchy instead. (Or they could even tap the already selected tab to jump to its top level and start over.) If you use a tab bar, it should provide the top-level navigation of the entire app. Tabbed views can contain navigation controllers, but not the other way around.

Deciding to go tabbed is a big choice, because that tab bar will be visible for most or all of the time people are using the app. You can offer only four or five top-level categories at a time. When you have more than five, the burden is on the user to decide which ones she thinks she'll need most often; everything else gets hidden behind the More tab. Even if you have only a few tabs now, this decision can come back to haunt you when you want to add more of them later. Unless your app strongly benefits from the always-there top-level navigation provided by a tab bar, you might want to consider an ordinary navigation controller instead.

Every screen, except for some special screens dedicated to specific tasks (modal views, described shortly), needs to dedicate 49 points of height to that heavy black bar. (And even though you can tint it any color, you should keep it relatively heavy.) The tab bar's size, color, and shiny highlighting effects give it a dominant role in the visual composition of a screen, compared with lesser elements like toolbars. That reinforces the significance of the tab bar's role as the top-level navigation of the whole app, with the power to send users to different branches of hierarchy with only one tap.

Segmented-Controls-as-Tabs

From time to time, you may want to offer a couple of views of the same information, or variants of the same screen. You might be able to offer a switch between those options in the form of a segmented control that behaves like tabs. A proper tab bar controls the whole app, jumping from one top-level section to another, but this lightweight tab style (let's call it segmented-controls-as-tabs) affects only the content or presentation of the current screen. As a result, you can offer two or three personalities for a single screen. If you have a popover to give quick access to more than one set of controls, such a segmented control may be the answer. iWork makes good use of this technique in its inspector popovers.

Don't rely too heavily on this technique. It can feel arbitrary and confusing when a navigation scheme switches back and forth between the horizontal sliding of a navigation controller and the control-swapping of segmented-controls-as-tabs.

Multiple Personalities

Here is a way to gracefully cram a couple of completely distinct interfaces and navigation structures into one app. In response to the tap of a button (usually in an upper corner of the screen), the entire interface transitions to reveal a new interface, sometimes with its own navigation scheme. The most prominent example is iBooks, with its entire store-shopping experience presented on the reverse side of its reading experience, complete with a three-dimensional flip transition between the two. The big, fancy animation makes it feel as if you're taking a major navigational leap—"going to the store"—without having to open a separate app. When you buy a book, it hovers in

midair as the app flips back to the bookshelf, thus logically connecting the two sides. It's a fairly rare scheme, and you should use it only when you're certain you need it, but it can be invaluable when you need to include two similar but distinctive interfaces in one app.

Modal View

You can use a **modal view** to handle a specific task that doesn't quite fit into your ordinary navigation hierarchy. While the modal view is open, the normal navigation and functionality of the app are temporarily unavailable; the app is in a specific "mode," hence the name. A classic example is composing a message in the Mail app. This mode is available from anywhere and has nothing to do with where you are in the hierarchy, so it takes you out of the hierarchy momentarily to deal with the task of writing a message. Then it drops you back where you were.

On iPad, you have several choices for presenting modal views (see Figure 3.5), each with its own personality.

- **Full screen**—The iPad screen is pretty big, so a full-screen modal view is a big deal. This option makes sense when users will be spending a lot of time in the mode and it's OK for them to forget about the main app itself in the meantime. A full-screen modal view takes over the whole device, behaving like an app within an app, dedicated to a specific task. Think of things like a web browser built into a Twitter app. You could very well follow a link that someone tweeted and end up spending 45 minutes reading an engrossing article or watching a video; that experience needs to use the whole screen, and not relegate you to a fragment of the screen while the rest of the app peeks out at you in the background.
- **Page sheet**—This is a step down from the full-screen style, in that a page sheet has a constrained width. In portrait orientation, it looks like a full-screen modal view, filling the width of the screen at 768 points wide; but in landscape orientation, it's still only 768 points wide, leaving some of the underlying interface visible but dimmed. The Mail app uses this style for its composition window. The resulting experience is close to the full-screen style. Users can easily spend a lot of time and thought writing an email, so they should be given a quiet, dedicated space to do so. But there is one key difference from the full-screen style: a page sheet keeps the interface from getting too wide. A 1,024-point-wide composition area would result in really long lines of text, a classic blunder of poor typography; the distance the eye needs to travel to go from the end of one line to the beginning of the next is too far, causing reading mistakes and cognitive fatigue. (See more about typography in Chapter 4.)
- **Form sheet**—This is a step down from a page sheet. A form sheet takes up only 540 × 620 points, hovering in the middle of the screen, with the rest of the interface dimmed. This doesn't depart very far from the normal context of the app, so it feels more lightweight. It's useful for tasks that need a bit of space but that you expect to take only a moment, such as entering account credentials for a web service, exporting a document, or changing appwide settings.

- **Current context**—Sometimes you need to present a mode *inside* an existing view, such as a popover or one side of a split view. Perhaps you want to allow access to the sidebar pane while a mode is under way in the content pane. Or you need to break out of the navigation hierarchy of a popover to take care of a special task. You shouldn't need to use this style of modal view very often, but it's good to know you have it for special situations.
- **Popover** (as an alternative to a modal view)—A popover isn't usually a modal view, but you can use it to fulfill the role of one in a lightweight way. When you're considering a modal view, ask yourself whether a popover might do the job better. A popover keeps users much more in context rather than pull them away from what they're working on. (If you need to make a popover feel more modal, you can give it Cancel and Done buttons and disallow tapping away to dismiss it, but this design is falling out of fashion.) So if the task at hand is related to the existing interface on the screen, and doesn't need all the space afforded by a modal view, try a popover instead.

On iPhone, modal views are simpler: they're always full-screen, because anything smaller would be barely worthwhile (although the **partial curl** transition style does leave much of the existing screen in place). Most of the time you should use the straightforward vertical transition, which simply slides the modal view up from the bottom of the screen and then slides it away when the task is finished. Notice that the transition doesn't push the previous screen out of the way, as horizontal navigation does; instead, it merely covers it up, thus promising that it'll still be there when the mode is over. Users recognize and understand that type of transition as being a momentary diversion from normal navigation. Other transitions might leave users guessing about whether they've been transported to some other part of the app. See Chapter 6, The Prototypes, for more about transitions.

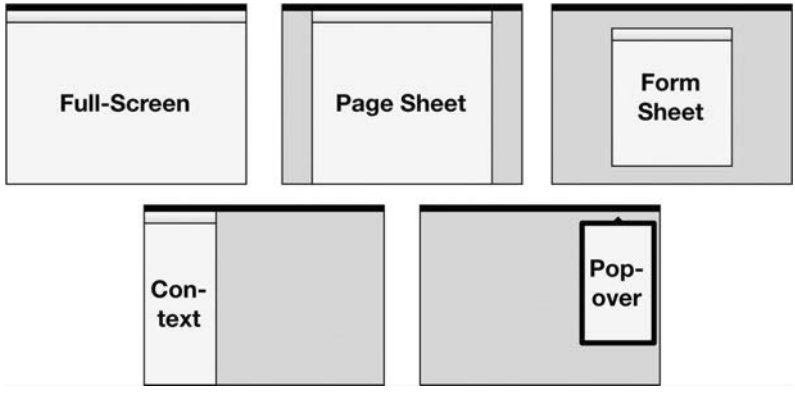


Figure 3.5 Types of modal view. The "current context" type can appear anywhere; in this example it appears in a sidebar. A popover can be modal or nonmodal.

Popover

A popover is a marvelous little iPad-specific element that seems unremarkable at first. It's a little window with a triangular stem pointing at the object that summoned it. But the humble popover contributes great sophistication to the iPad experience, for a few subtle reasons.

- You don't need to manage it like a window on the desktop. It can't be moved, and it's generally only as big as it needs to be.
- It appears at the time and place you need it.
- It disappears as soon as you're finished with it, either because you reached the end of its little workflow or because you decided to stop using it and tapped away.
- It generally keeps you in the context of the surrounding interface so that the interlude of using the popover doesn't interfere too much with the train of thought you had before opening it.
- It assumes that you're editing in-place and that you want your work to be saved, unless you manually cancel or undo it. Simply tapping away from a popover shouldn't discard your charges.

All this combines to make using popovers a lightweight experience. Consequently, interfaces can be designed to be "quieter" than they would be otherwise. Functionality can be easily summoned, used, and then put away for next time, rather than spread out all over the screen all the time so that you can get at it easily. (See Chapter 14 for more about quiet interfaces.)

Another exciting thing about popovers is that they can contain screens and navigation hierarchies all their own, distinct from what's going on in the main app. You can think of a popover as a little iPhone sitting inside the iPad screen, with its own miniature app that deals with one concern of the greater app. Popover navigation can use navigation controllers, segmented-controls-as-tabs, and modal views.

Custom Navigation

So far, we've looked at the standard navigation methods offered out of the box with iOS. If you need to—and if you have the software engineering wherewithal at your disposal—you can create almost any navigation scheme that you dream up. But be careful. Lots of apps with unique needs present their experience in a clever, original way, but not all apps have those unique needs. Applying an unexpectedly distinctive design to a problem that could have been solved with a standard approach can backfire. Think carefully before deciding to use a clever and unique navigation scheme as a primary way of differentiating your app. (See Chapter 14 for advice about unique designs.)

The most important thing to keep in mind when thinking up new navigation schemes is that they should feel spatially consistent. The conventional navigation controller scheme works well in users' minds because they can easily build a map of how

it works (often called a **mental model.**) Deeper levels are further to the right, higher levels are off to the left, and modal views slide in from off-screen and then slide away when they're no longer needed. In iBooks, tapping a book on the shelf causes it to zoom toward the viewer and open. The iWork apps present a similar experience by showing a grid of documents that zoom up when you tap on them and zoom back when you're finished editing. GarageBand is more ambitious. It starts with that same document grid but adds a carousel of various instruments, and a tracks overview is revealed by a vertical rotation effect. (See Figure 3.6 for a map of GarageBand's navigation scheme.)

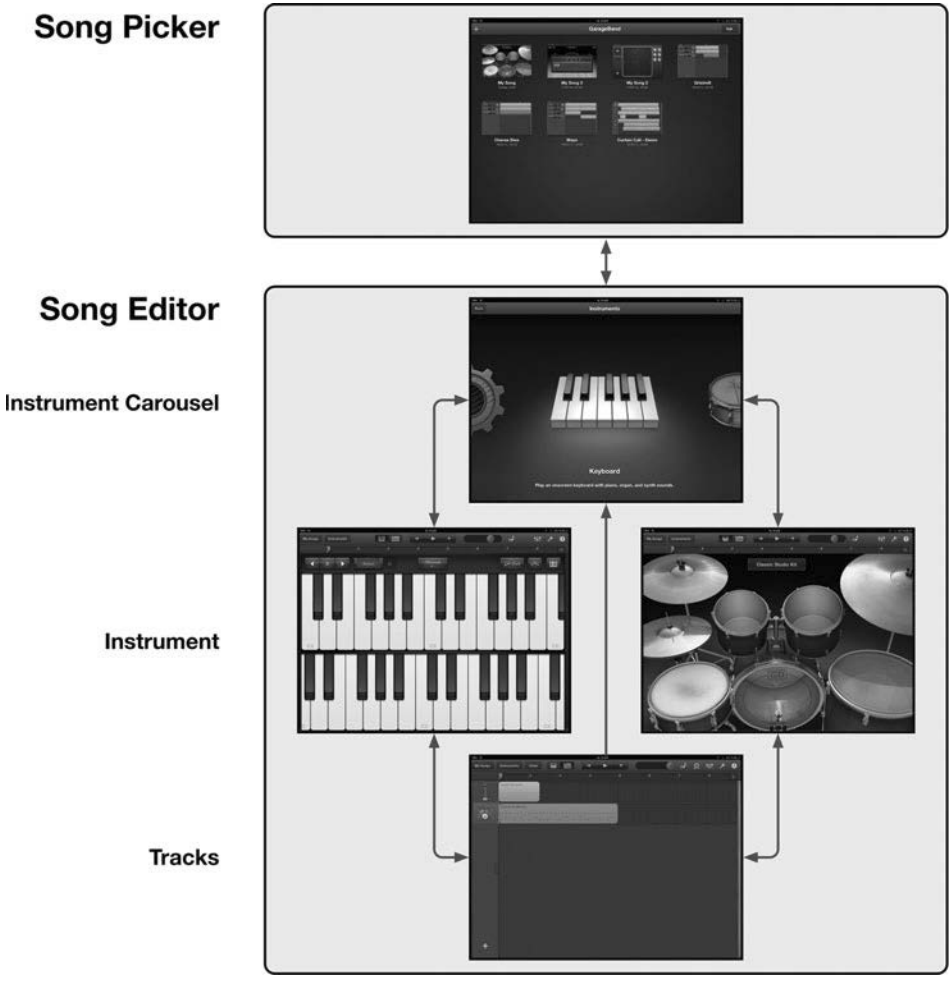


Figure 3.6 GarageBand on iPad has a custom navigation scheme that's complex but consistent.

Even if users never consciously think about the mental map they've made of the app, they'll be put off if that mental map is violated. Use smooth, sensible transitions every time the screen changes, especially when the changes are big ones. The simpler you can make the spatial representation of your navigation, the easier it will be to make it feel stable and consistent to users.

Advice on the Standard Elements

Elements are the building blocks that make up screens: views, controls, alerts, and so on. In wireframing, you just need to find the right elements and put them together on the right screens in the right arrangement. Of course, that is like saying that writing a best-selling novel is only a matter of picking the right words and putting them in the right order. You'll need a lot of wisdom to put together great app designs, but to begin with you should make sure you're familiar with the building blocks at your disposal.

As a rule, you should choose standard elements provided by the OS over building your own. For nearly every need, there's a standard control that does the job reliably and predictably. Standard controls have the benefit of being familiar to users, who of course spend most of their time in other apps. Chapter 14, Consistency and Specialization, goes into more depth about the choice to go standard or custom.

For a basic understanding of the standard elements the platform offers, you should read the *iOS Human Interface Guidelines*. The beginning of the chapter said that, too, and so did Chapter 2. Really, you should read them. That will give you Apple's official baseline position on how these elements are to be used. But for further advice and suggestions about how to put them to use effectively, based on actual Apple and third-party examples, here is a rundown of each individual element.

Bars

These basic screen-spanning bars show content and controls.

- **Status bar**—The only decision you need to make about the status bar is whether to hide it and, on the iPhone, what style it should be. The different states of the status bar mainly affect the immersiveness of the app; the more immersive treatments avoid distracting the user from the content being shown. The standard light status bar blends in with the interface. This works best when there's no immersion necessary, as in communication or productivity apps. On the opposite end of the spectrum are media or entertainment apps, where you'll want to hide the status bar to avoid distracting from the content. But don't hide the bar just to show off how cool and immersive you can be. Users tend to care about the time and their battery level. Only if the immersiveness is more important than that information can you justify hiding the status bar. If you're not hiding the bar, remember to accommodate 20 points at the top of each screen in your wireframes and mockups, and consider what your screens will do when the double-height status bar appears. (See The Worst-Case Height Compression Scenario in Chapter 4.)

- **Navigation bar**—This bar spans the entire top of the screen (on iPhone) or a specific view (on iPad). It's 44 points tall, except in landscape mode on an iPhone; then it's 32. A navigation bar is the primary way people move through the various screens. It sits at the top of the view, framing it with a reminder of where you are and how to go back up a level. Don't confuse the navigation bar with a toolbar. Navigation bars are only about showing a back button, a title, and possibly a single (bordered-style) button for managing content (an Add button, an Edit button, a View button, etc.).
- **Toolbar**—This bar spans the entire screen (on iPhone) or a specific view (on iPad) and serves to contain controls. It's 44 points tall, except in landscape mode on an iPhone; then it's 32. If the navigation controller is the workhorse for moving between screens, the toolbar is the equivalent for choosing commands on screens. A toolbar can contain a limited number of controls, so you'll need to be scrupulous about picking them. On iPhone, you're limited to five 44×44-point buttons. On iPad there isn't a hard limit, but you should do your best not to clutter the bar. Unlike desktop computer users, iOS users aren't accustomed to wading through lots of controls. Instead you should think of ways to consolidate functionality behind a single button, by using popovers, modal views, and action sheets. (See Chapter 14.)

  You have two choices of button style in a toolbar: regular and bordered. The bordered style is great for emphasizing that something is a button, when going borderless would make it ambiguous. A button with a text label is the prime example of something that needs an extra bit of emphasis. Another example is when there's non-buttony stuff hanging around in the same bar, and that's why a button in a navigation bar is always bordered; otherwise, it could be construed as being part of the screen title. The HIG advises against using both button styles in the same toolbar, but on iPad such a design isn't a big deal. iWork keeps bordered buttons on the left side of the toolbar, and borderless ones on the right. That's fine; just don't mix the styles willy-nilly. (See Figure 3.7 for the example set by Keynote's toolbar.)
- **Tab bar**—This is a 49-point-tall bar that spans the entire screen and always appears below the content it switches. (The purpose of a tab bar is described in detail earlier in this chapter, in the section called Tabs.) When you're

Figure 3.7 Keynote has similar toolbar configurations on iPad and iPhone. Bordered and unbordered buttons are kept separate for the sake of tidiness.

wireframing for tab bars, remember they're slightly taller than toolbars, and you can put only five tabs into one on iPhone. For a while, it was fashionable to make a custom tab bar that promotes one tab by having it protrude outward from the top of the bar. Instagram made that design famous. But the practice seems to be falling out of favor and doesn't appear in that app anymore. Fads.

Content Views

These are the basic, general-purpose views for presenting content and controls.

- **Popover**—Earlier you learned why popovers are awesome, so here are a few tips for keeping them that way. For the most part, users expect popovers to be 320 points wide, for that iPhone-within-an-iPad feeling. Popovers much wider than that feel awkward hanging off that tiny triangular stem, and a form sheet would probably work better. Make popovers as tall as you need, of course, but no taller.

  When navigating among the screens of a popover, don't worry if you leave empty space (because the popover is too tall for the content) or if scrolling is required (because it is too short). Such minor imperfections are better than making users sit through a resize animation every time they navigate somewhere. So just find a height that works reasonably well for all screens in the popover, and stick to it.
- **Split view**—Remember that the sidebar is always 320 points wide, mimicking the width of an iPhone screen and allowing you to use similar layout strategies for both sidebars and iPhone screens. Normally, the sidebar is always visible in landscape orientation; in portrait view, it is often hidden and needs to be slid in from the side. But thanks to the official Facebook app, it's fashionable to offer a slide-in sidebar that is always hidden until summoned, regardless of the orientation. This design puts more focus on the content area, at the expense of quick access to navigation. The tradeoff is that it's a bit nonstandard and thus is harder to implement and maintain. Weigh that against the importance of emphasizing the content in both orientations.
- **Table view**—Table views are the go-to element for displaying information, editable or otherwise. Table views appear in sidebars, main content areas, popovers, modal views…everywhere. They can be used for navigation (with chevrons or detail disclosure buttons), for selection (with checkmarks), or for data editing (usually using the value styles, described shortly). The name is a bit misleading: you might expect a table to offer multiple columns, but iOS table views are a single column (although each cell can display several bits of information). There are two standard table styles and four standard cell styles to choose from, each with a terribly nondescript name that makes it hard to keep them straight.
- **Text view**—Standard text views are straightforward. They're good when you need to display, or the user needs to enter, lots of text. Of course, "lots" is relative.
- **Web view**—There is about one good reason to use a web view: to load an actual page from the web for tasks like logging in to sites or following a link in the user's content. You formerly had to use web views to display rich text with

styling applied to it, because the standard text view was plain-text only; but as of iOS 6, you can put rich text in labels and text views. (Some apps try to put most of their functionality inside the web view, actually running a web app wrapped thinly in an iOS app. For advice on that particular venture, see the hybrid web-app admonishment in Chapter 14.)

Here are the styles of cells you can use in your table views.

- **Plain table**—This table style pushes the content all the way to the edges of the view; an example is the message list in Mail. A plain table is great for presenting a single, homogenous list of items, especially when the list is likely to get long. That's why you see it used for email messages, contact names, music tracks, to-do list items, or any other uniform collection of items that can get arbitrarily long. Even within such homogenous lists, though, there can be subdivision indicators, such as letters of the alphabet. In those cases you can use **section headers,** which float among the rows as little eye-catching milestones without being tappable. And if a sectioned list is likely to be *really* long, you can provide an index along the edge of the screen for quickly jumping to a certain group. The index tends to look silly or awkward if you don't have lots of data or if you use it for sections that aren't strictly ordered in a familiar way such as the alphabet.
- **Grouped table**—This table style keeps the content in self-contained stacks with rounded corners. An example is the Settings app in iPhone. A grouped table works best when you have heterogeneous kinds of information to present on one screen, when the separation of sections is critical, or when you offer controls and labels that don't fit into table cells. Imagine if the Settings app had a single huge plain-style table for all the settings. There would be no strong separation between the kinds of options; there would be no place to put the explanatory labels; and lots of sections would consist of a contrived header and a single item. See Table 3.1 for a comparison of plain and grouped tables.

Table 3.1 **Plain Versus Grouped Table Views**

Plain Table	Grouped Table
It's good for any length of list.	It's best for shorter lists.
It's good for homogenous data.	It works fine with wildly different kinds of data.
Labeled sections help delineate subgroupings like alphabetization.	Groups provide stronger separation than sections.
All sections need to be labeled.	Groups can be labeled or unlabeled.
All sections are expected to hold roughly equivalent kinds of data.	Each group can hold a different kind of data.
It can offer an index for jumping quickly to a section.	A single index doesn't make sense, because there is more than one table on the screen.

- **Default cells**—The default cell style simply presents a text label and, optionally, an image. This style is generally used in navigation to help users identify the information they want to navigate to next. It's used to identify artists in the Music app, people in the Contacts app, and so on. The designers could have used the subtitle style to include plenty of extra data, but in these cases they recognized that almost every time, users want to navigate to the full detail screen. No single piece of information would be most helpful to promote from the detail screen to the cell, so they left the cell simple, encouraging users to delve into the detail screen. After all, taps are cheap, and it costs very little to delve inside and get a dedicated screen of information about the item you're interested in.
- **Subtitle cells**—This cell style includes smaller, gray text below the title to give additional detail about each item. It's useful when you need to let users compare a certain key piece of information between items without having to delve into the individual detail screens. For instance, the app list in the Notifications screen of Settings uses a subtitle to indicate the kinds of notifications set up for each app. Users can see at a glance the answer to their likely question: "Which apps have notifications enabled, and what kinds?" It's a clear case of a piece of information that saves people time and trouble when it's pulled up a level into the cell.
- **Value 1 cells**—Yep, that's the official name of this style. In the Interface Builder component of Xcode, it's called "right detail," which is a bit more descriptive. This style combines a label and a value, usually to allow users to edit the value. The label is bold and left-aligned; the value is grayish-blue and right-aligned. This layout emphasizes the label text, because (at least in left-to-right languages), people find it easier to compare strings of text that are left-aligned; scanning down the beginnings of the labels is easy. You can also offer switches or other small controls in the value area instead of a text string. Value 1 cells are seen in various places throughout the Settings app, accommodating users' need to scan the names of the various options presented there, looking for the one they're interested in. Scanning the values wouldn't help that process very much.
- **Value 2 cells**—Also known as "left detail," this style combines a label with a value that can usually be edited. The text label is deemphasized by its small size and its fainter color, whereas the value is dark and bold. In this layout, it's easy to recognize and compare the values, because they're left-aligned in the middle of the cell. This cell type works well when you have pieces of information that are more recognizable by their values than by their labels, and that's why it is used in the Contacts app on iPhone. Names, phone numbers, addresses, email addresses, and other bits of contact information are pretty recognizable without seeing the label, so you can easily scan the values on the screen and find the one you need. (See Figure 3.8 for a comparison of the various cell styles.)

Figure 3.8 Left to right: default, subtitle, value 1, and value 2 cells. Each is useful for listing information in a subtly different way.

- **Custom cell styles**—You can put almost anything in a custom table cell. So if you need something a bit different (or very different) from the standard choices, and you have a good reason for it, you can create your own style. For an example, see the message list in Mail. Each row provides a large, bold sender name, a small, regular-weight subject line, a small gray message preview, and a blue time indicator. When you dig through your email looking for a specific message, all these bits of information are likely to be helpful; none of them could be omitted safely. To implement the message list using one of the standard styles would have done a disservice to the task of browsing email messages. (See Figure 3.9 for some custom table cell styles.)

Alerts

Alerts are an efficient, effortless way to drive your user crazy. For decades on the desktop and on the web, the alert was a way to pester users with whatever the developer felt like saying, whether the user cared or not. iOS did a lot to reduce the number of alerts that users have to deal with, reserving them for times when something is seriously amiss and the user needs to be notified or asked for immediate input. Consequently, when an alert appears it usually heralds an important moment. If it instead says something inane, banal, or cryptic, it waters down (or poisons) the purpose of alerts and reduces the attention users give them.

Here are some good times to use alerts:

- When the app can't proceed unless users enters their account credentials
- When a background process encounters a problem, such as a sync conflict, that needs an immediate decision from the user

Figure 3.9 Left to right: Mail, Podcasts, Tweetbot, and Instapaper. Custom table cell styles support any layout you can dream up.

The following are some terrible times to use alerts.

- When an operation completes normally. The interface should make this apparent without interrupting the user.
- When something the user just tried to interact with, still visible on the screen, has a problem. In that case, the thing itself should show its status.
- When you want to influence the user's behavior (see Chapter 10).

Look at the animation that a new alert performs when it appears on the screen. It emerges from nothing, fades in, floats perpendicular to the screen toward the user, and then bounces back into place. Almost everything else that happens on an iOS device is connected to something already on the screen, giving it context and meaning. An alert is for those rare cases when something happening behind the scenes, something that normally proceeds without the user's needing to care about it, suddenly needs the user to care about it.

Action Sheets

Action sheets are another humble but heroic player in iOS's quest to make software quieter and more respectful. Action sheets are a stack of buttons that appears in response to an action by the user. Their effect is profound.

On iPhone, action sheets always slide up from the bottom of the screen and always offer a cancel button in addition to their action buttons. On iPad, action sheets might do the same thing in the virtual-iPhone-interface that is a popover, or they might appear in a dedicated popover of their own. If they appear in a popover of their own, there is no cancel button; instead, the rest of the screen serves as a safe place to tap away and cancel the action.

Here's what is cool about using an action sheet.

- It hides several similar actions behind a single button, thus simplifying the interface until the moment of need.
- It doesn't necessarily offer explanatory text, thus making it feel lightweight and reinforcing the fact that it always appears because of the last thing the user did. (You can add explanatory text if you really can't get the situation across with the button labels.)
- It's easy for a user to open, check the available options, and then cancel if none of them is what the user wanted.

An especially thoughtful case is the action sheet that shows a single action button. In old-fashioned interfaces, where an especially consequential action would be accompanied by a fussy dialog box with long explanatory text and buttons for proceeding or canceling, iOS offers a single, clearly labeled confirmation button. If you really intend to proceed, you can move your finger an inch and tap the action button. If not, you just tap away. That's elegant.

Standard Controls

Most controls are fairly straightforward and are well documented in the HIG. Here are some tips for using them.

- **Activity indicator**—This is also known as the **indeterminate progress indicator,** or **spinny.** On iOS, these are much more common than progress bars. If something takes less than a few seconds, you just put up a spinny in a location connected to the work being done, and don't bother the user with guesses about how much time is remaining. An activity indicator should suggest what it's for by an associated text label, by its location on the screen, or both.
- **Date and time picker**—This is also known as the **wheels of time.** Nine times out of ten, this is the right way to get date input from users. Spinning to dates that are even decades away is quick and easy. If multiple fields are visible on the screen, this control helps highlight or otherwise call out the value that the wheels are editing.
- **Detail disclosure button**—Normally, when you need to delve inside an item to see more detail on another screen, you tap a table cell that has a chevron on the right side. A detail disclosure button serves as a backup "delve inside" tap target for times when you can't follow that pattern. You might need to use it in two cases.
  - The item to be delved inside isn't a table cell and thus isn't obviously tappable for more information—for example, the bubble that emerges from a pin on the Maps app for iPhone, or a photo in Messages.
  - The table cell itself has some other function. In the Phone app for iPhone, tapping the cell for a favorite contact starts calling the person, whereas tapping the detail disclosure button delves into a detail screen about the person.

  In that second case, adding a detail disclosure button makes the most sense when both choices are about equally likely and when the shiny blue button doesn't compromise the cleanliness of your visual design. Another option is to split the two functions between normal mode and Edit mode.
- **Info button**—This venerable emblem, which is used on the desktop mainly for editing content details, is supposedly for revealing "configuration details" on iOS. For a while the same icon was used in the iWork apps on iOS to summon the style inspector popovers, but it has been replaced by a more expressive paintbrush icon. Meanwhile, the tools popover, which is closer to "configuration details," is summoned from a *wrench* icon. Many third-party apps that have configuration screens opt instead for a gear icon, because the meaning of the info icon is so muddled.
- **Label**—This is an ordinary little string of text that you can use to…label things. Generally, it's best to match the style and layout of the default labels on a grouped table view.

  - To name something, put a bold label immediately (10–12 optical points) above it. Keep it on one line.
  - To offer additional explanation about something, put a regular-weight label immediately (10–12 optical points) below it.
  - To offer freestanding explanatory text that's not related to a particular element, put some empty space (20–24 optical points) between it and the nearest controls.

  You don't need to label everything. A group of obviously color-related controls doesn't need to be called "Colors." The only table on a screen titled "Addresses" doesn't also need to be labeled "Addresses." Make sure you add labels only when they actually communicate something that wouldn't get across otherwise.

- **Network activity indicator**—This activity indicator in the status bar informs the user of communication happening over the network. Users look here to see whether their network connection is being used, especially if they're expecting some stale information on the screen to be updated. This indicator is a subtle hint to keep waiting, because the update is on the way.
- **Page indicator**—This is yet another quietly heroic interface element. Thanks to its presence on the home screen, this simple series of dots is immediately recognizable to most users as an invitation to swipe sideways for more content. It gives you the opportunity to display lots of screen-sized chunks of information without actually requiring navigation from screen to screen. (See Chapter 12 for more praise of pagination.)
- **Picker**—This generalized variant of the wheels of time is used for pop-up menus on web sites but is pretty rarely seen natively. Most times that you would use it, you could instead use a table view; the interactions of scrolling through a table view and of spinning the wheel are almost identical. The main benefit of the picker is that it lets you stay in context, and that's why it works well on the web. (On a web site, you need to stay in context in order to see the identifying information around the pop-up, so you can't just navigate to a dedicated screen for a table view. Nor can you insert an arbitrarily tall table view into a web site that wasn't designed for it.) Another benefit is that it's slightly lighter weight than a table, because the user simply scrolls to update the value; there's no need to tap an item to select it.
- **Progress view**—This is equivalent to the old-fashioned progress bar seen often on the desktop. As mentioned in the description of the activity indicator, most of the time you don't need a progress view. First, most operations should not take so long that you need to show the user how far along they are. Second, people need to see a progress bar only when they have no choice except to wait for the process to complete in order to get something done. Here are good examples of using progress views:
  - Waiting for a document to be downloaded from iCloud so that you can work on it

  - Waiting for an iMovie project to be exported so that you can send it to someone
  - Waiting for an email with heavy attachments to be sent so that you can make sure it succeeds

  If the process usually takes less than a few seconds, or if waiting for it doesn't affect the user's ability to get work done, you're probably better off with a spinny activity indicator.

- **Rounded rectangle button**—This is the one general-purpose, standard style of button that you can place in content areas. When you're using buttons in the content area, make sure you use them for *actions*. Don't use them for the following other purposes (as always, unless you have a good reason to).
  - Navigation is usually better handled with table cells bearing chevrons or detail disclosure buttons.
  - Choosing from a number of options is usually better handled by a table view with checkmarks or a segmented control.
  - Toggling a setting on or off is usually better handled by a switch.
- **Search bar and scope bar**—These are handy when a screen shows a number of items and it takes more than a few moments to scroll through them and find one manually. A common trick is to include the search bar at the top of the content area and load the screen so that it's scrolled just out of view. This design lets people scroll to the search bar if they need it but otherwise leaves it tucked away out of sight.
- **Segmented control**—A segmented control is a concise way to offer a very short list of mutually exclusive options. Often, it's a headache to come up with good labels that fit inside the narrow buttons, and you should use a table view with checkmark selection instead. The following are some good uses for segmented controls.
  - Selecting from a handful of options, if you can get your point across with recognizable images or very short text labels. You can even offer a label just above or below the segmented control that updates to reinforce the current choice with a text description.
  - Providing options that show or hide other controls based on the setting. The visual weight (see Chapter 4) and experience weight (see Chapter 13) of the content-area style of segmented control lend it well to this use. Pushing a big segment and watching it highlight in intense blue feels appropriately consequential to the subsequent appearance or disappearance of controls.
  - Switching between views on a screen in the same way tabs do. See Segmented-Controls–as-Tabs earlier in this chapter.

  Whatever you do, don't make a segmented control behave like a button. It's for choosing between options, and not for performing actions. And remember that

for simple on/off toggles, you have the switch at your disposal; a segmented control with "on" and "off" segments doesn't make a lot of sense.

- **Slider**—This is a great way to provide quick control over a continuous setting when the actual numbers aren't very important. Excellent examples in the operating system are the brightness and volume sliders. Nobody ever thinks, "I could hear better if the volume was at 86% right now" or, "It's getting dark; I should turn the brightness to 39%." Instead, they think, "Quite a bit louder" or, "A little dimmer." That's the sort of thing sliders excel at. People don't know exactly what setting they need beforehand; instead, they need continuous feedback while moving the knob. So make sure users can see or hear the result as they move the slider. It's frustrating having to go somewhere or do something to get the feedback they need.
- **Stepper**—A stepper is good for numerical settings when the number matters but adjustments tend to be within a small range. Poking the plus or minus button until you see the number you want is a cognitively cheap interaction, compared with typing in a number. A setting that was almost always set to 1, 2, or 3, for instance, would work well with a stepper.
- **Switch**—You can put a switch in a table cell to offer a simple on/off toggle. Flipping a switch feels fairly weighty, so you can easily use it for consequential settings or let it show or hide other controls. Make sure that the two opposing settings are easily gleaned from looking at the label. Something like "Automatically download new items" goes well with a switch. A vague title like "Horizontal Layout," where the opposing option is not immediately apparent, is not as good. (That would be better served by a segmented control with the label "Layout" and segments called "Horizontal" and "Vertical.")
- **Text field**—The text field is commonplace on the desktop and the web, but it often feels a bit crusty on iOS, especially if it's just sitting in a content area, lacking any placeholder text. Sometimes, you can offer a better way of entering information, such as picking from a table view. Text input is even more of a pain on a touchscreen than on a physical keyboard, so avoid it when you can. In content areas, a table cell with text input enabled is often more attractive. But when you need text input and you can't use a table cell, then a text field makes sense. (See a comparison between text fields and table cells in Figure 3.10.)

Figure 3.10 Text fields (left) look dated, feel cramped, and are hard to balance. A table cell with text input enabled looks nicer, and using placeholder text instead of a value cell style leaves more room for typing.

Custom Controls

If you work on a sufficiently complex software project for a long enough time, you're bound to run into cases where the standard building blocks don't quite provide the best experience you can imagine. This section isn't about customizing the appearance of standard controls; that's a topic for the mockups phase in Chapter 5. Instead, it's about making controls that behave in a new way.

Most of the time, you can and should base your custom control on a standard one. There's likely to be a standard control that kinda does what you want but that you need to tweak. By carefully adjusting its characteristics to satisfy the interaction you have in mind, you can keep your custom control as close as possible to the spirit of the platform.

Of course, it's possible to create an entirely original control from whole cloth, without basing it on anything that came before. If you and your team can pull this off perfectly, you'll be heralded as UI design heroes. If you execute it anything less than perfectly, however, it'll come off as awkward and painful to use.

Suppose you want to provide a long list of options in a quick, easily browsed way, without taking up a lot of space. These options are easily represented by small square icons, so a big tall table view with labels would be overkill. But a segmented control can't hold all the options you're planning. What kind of custom control could you create to do the job?

Start by looking at the standard controls that offer a way to pick from a list. The picker control's vertical spinning wheels do a good job of offering a lot of options without taking up much space. What if you could adapt the picker concept to a more compact, icon-based set of options? Presenting…the **horizontal mini-picker.** (See Figure 3.11.)

This control can hold any number of options, as long as they're represented by distinctive, square icons. It takes up only as much space as a single table cell row, thanks to its horizontal orientation. And it's immediately familiar, because it takes advantage of an existing control's interaction metaphor. In fact, to typical users, it probably wouldn't be obvious that the control was custom made. For all they know, it's simply a standard control offered by the operating system. That's great! Blending in as a natural part of the platform is a noble goal.

There are plenty of ways a control can give away its custom status. If you miss any of the following considerations, your control is a lot less likely to be perceived as fitting in with the rest of the system.

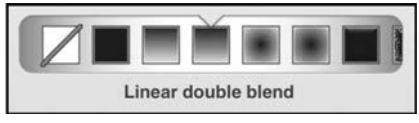


Figure 3.11 A horizontal mini-picker for choosing fill styles in OmniGraffle. It tweaks the concept of a vertical picker for a slightly different purpose.

- How does the control react to various gestures—tap, double-tap, touch and hold, drag/swipe, and the like?
- What happens if you accidentally touch the control and then try to drag your finger away without letting go? (On a standard button, this lets you cancel a mistap.)
- How does the control adapt to different amounts of available space, especially when the orientation of the device changes?
- What does the highlight look like while you're in the middle of tapping the control?
- How does the control work with accessibility features, especially VoiceOver?

Summary

iOS offers a healthy collection of carefully thought-out building blocks that you can use to craft your own navigation hierarchies and screens. Plenty of life-improving apps can be built using only these standard elements and navigation schemes. But if you need to, you can build your own custom navigation and custom controls. Just make sure that anything you create conforms to the spirit of the platform.

Now that you've familiarized yourself with the toolkit available (by reading the HIG and this chapter), you're ready to start building wireframes in earnest.

Exercises

It's time to try out your new knowledge. Give these exercises a shot to solidify your familiarity with the standard iOS elements and your understanding of when to customize beyond them. Do each one a few times, choosing a different example for each iteration, if you like.

1. Think of a single feature in your own app. What screens and elements might you need? Sketch out a couple of approaches using different kinds of controls to see which one feels right.
2. Choose a standard control. Imagine how you could design a custom version of it that serves a slightly different purpose. Can you make it more precise (or less, if that's what is needed), more compact, or more expressive? What purpose would your custom control serve better than any existing standard control?
3. Draw the geography of your app the way GarageBand's is drawn in Figure 3.6. Can you make spatial sense of the navigation scheme you're using?

This page intentionally left blank

Index

C

D

E

F

G

H

I

J

K

L

M

P

Q

R

S

T

U

V