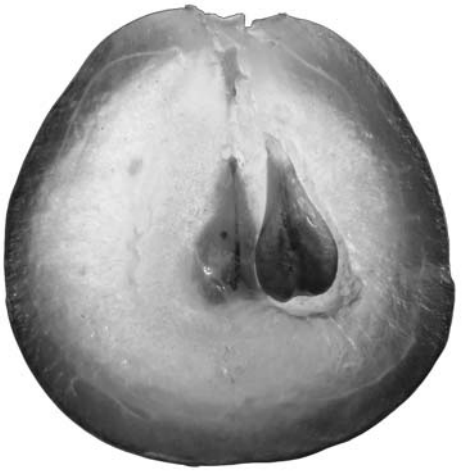
iPhone User Interface Design Projects



Dave Mark, Series Editor

David Barnard
Joachim Bondo
Dan Burcaw
David Kaneda
Craig Kemper
Tim Novikoff
Chris Parrish and Brad Ellis
Keith Peters
Jürgen Siebert
Eddie Wilson

Apress®

iPHONE USER INTERFACE DESIGN PROJECTS

ISBN-13 (pbk): 978-1-4302-2359-7

ISBN-13 (electronic): 978-1-4302-2360-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

President and Publisher: Paul Manning
Lead Editor: Clay Andres Matthew Moodie
Developmental Editor: Matthew Moodie
Lead Author and Technical Reviewer: Joachim Bondo
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Michelle Lowman, Matthew Moodie, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Coordinating Editor: Kelly Moritz
Copy Editor: Heather Lang
Compositor: MacPS, LLC
Indexer: John Collin
Artist: April Milne
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please e-mail info@apress.com, or visit http://www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at http://www.apress.com/info/bulksales.

To my wife and family for being so incredibly supportive
—David Barnard

To my wife, Malena, who once again gave me the support I hadn't earned
—Joachim Bondo

To Mom, Dad, and Zach
—Dan Burcaw

To Lindi and our kids, for being cool, hip, and weird. And to Brandy, for always keeping me on task.
— Craig Kemper

To all the people who helped me get started and to Oana for supporting me all the way.
—Tim Novikoff

To my family, Liz, Sovereignty, and Aidan; thanks for all the patience and support
—Chris Parrish

To all my colleagues of the FontShop network
—Jürgen Siebert

To my wife Jenna and daughters Cana and Mia
—Eddie Wilson

Contents at a Glance

Contents

Foreword

Dear Readers,

When we hatched the idea for a series of project-oriented books featuring the work of leading iPhone developers and their apps, there were very few people we could really turn to as recognized experts in the field. We were all beginners of one sort or another: first using Objective-C, first trying out Xcode, learning to write for a mobile device, or perhaps developing our first app for fun. Whatever differentiating baggage we each brought with us, we were sharing the experience of learning Cocoa Touch, the iPhone OS SDK, and an enthusiasm for this new thing.

About a year later, it's a much more sophisticated iPhone world that is receiving this fourth in the iPhone Projects series published by Apress. Not only are there more and better apps but there are many more experienced, truly creative developers and designers. And since interface design and usability become more important as differentiating factors for the most successful apps, we're featuring some of the most creative designers in this book.

iPhone User Interface Design Projects is unique within the series for being design, rather than code, focused. All of those hard-core developer topics that dominated our earlier books needed to be written, because there really is no other place to start. But the really successful apps, the ones you never get tired of using and remain popular on Apple's iTunes App Store for a long time, have great code and great design.

It's not enough to have a unique feature or great performance. Too many other apps will either be copying your unique feature or came out a week before yours. You've got to make great-looking apps, and this book has some key examples of putting a professional polish on your work. But it's really more than applying a simple shine, because as you'll see in these chapters, good design requires plenty of thought right from the start of the development process. And that may be the most important lesson you'll get from this book. You have to think about every aspect of your app if you expect to be one of the shining lights among over 100,000 apps.

Once again, I have worked with Dave Mark, the series editor and author of several best-selling Apress books, including *Beginning iPhone 3 Development*, to find developers who produce efficient and bug-free code, design usable and attractive interfaces, and push the limits of the technology. Dave's common-man touch, tell-it-like-it-is sense of reality, and delight at apps that look great can be felt throughout the series.

This brings us back to the code, or in the case of user interface design, the lack of code. As developers, we all take comfort in the language of code. This book is about the visual presentation of your code. Most of your users have no idea what beautiful code is, but every user can take pleasure when you present the inner workings of your endeavors as a truly beautiful, useful app. It's something every iPhone developer should show off with pride.

We hope you'll find the apps presented in these chapters and the stories of how they came to be interesting as both human drama and as well-designed as the iPhone and iPod Touch technology. Happy adventuring, and send us a postcard!

Clay Andres
Apress Acquisitions Editor, iPhone and Mac OS X
clayandres@apress.com

About the Technical Reviewer

Joachim Bondo has developed software for three decades, from programmable calculators in the late '70s before computers were commonly available to now the iPhone.

After releasing Deep Green, his critically acclaimed chess application, on the App Store, Joachim has contributed his excellent taste and insight on good user interface design to several Apress titles: *iPhone Games Projects*, *iPhone Advanced Projects*, and now *iPhone UI Projects*.

Introduction

By the time you read this, the number applications on the App Store will have crossed the 100,000 mark. Chances are that if you come up with an idea for an app, it's already on the Store and in abundance. In order to catch users' interest, you'll have to differentiate yours.

Most developers seem to take the easy route by simply lowering their price. The problem is that it's so easy to do that everybody can do it. And if they do, you've gained nothing. On the contrary, you've lost an income that could possibly motivate you to keep improving and updating your app and thereby sustain your name as a reliable developer users can trust their investment to.

If you choose to take the difficult route, however, and differentiate on quality—the route that not everybody can take—you'll not only become a better developer but also earn the respect from other developers and users who'll be willing to spend real time and money on your app.

The ten authors of this book have all released successful apps and can testify to how going that extra mile, or ten, has paid off in many aspects of their lives as iPhone developers. By reading their takes on how to make an app stand out from the rest, you'll gain some of the inspiration and insight that could make your app the one in its category that users will want.

If you're not going to differentiate your app on quality, this book is probably not for you. Instead, just go to iTunes Connect, and lower your price.

What's in This Book

This is a book about user interface design. As a consequence, you'll find lots of screenshots and only very little code. Several of the authors don't even have a programming background, but they all share the same passion for the iPhone and for developing apps of the highest standards in terms of user experience.

David Barnard of App Cubby is one such person; he has created a suite of essential utilities that enjoy great popularity on the App Store. In Chapter 1, he takes you through the process of perfecting entry views and presenting data, which both play central roles in his apps. And he explains how learning from Apple's UI conventions and usability testing can improve your final result.

In Chapter 2, I make an attempt to enhance the user experience and power of the navigation bar and present a relevant subset of large amounts of data in an exciting way. I bring you along on the same journey I took myself, as I actually came up with the design while I was writing the chapter.

Former Apple employee and Yellow Dog Linux distributor, Dan Burcaw talks on going native with his social networking app, Brightkite, in Chapter 3. He covers how that move made it possible to add the extra umph with CoreLocation, Camera, and Address Book integration that web apps just don't have. And, equally importantly, he explains how he tailored the user interface to match his target group.

Chapter 4 is devoted to Web and iPhone developer, David Kaneda, and the creation of his Basecamp project management client, Outpost. He starts with no idea and a blank piece of paper, continues over various attempts to creating the central dashboard screen, and finally settles on a version that manages to present a lot of information in a clear way.

Craig Kemper doesn't leave out any details when telling the tale about how he created his two award-winning puzzle games, TanZen and Zentomino. After reading Chapter 5, you'll understand why the two apps have been downloaded more than 150,000 times combined.

In Chapter 6 Tim Novikoff, a graduate student in applied math with no prior software programming experience, goes through his methodical process of creating Flash of Genius: SAT Vocab, an app for learning vocabulary words that made it to the Top 20 Paid Educational Apps list. His chapter is a fine example of how much you need to consider even when developing a seemingly simple user interface.

Long-time Mac developer, Chris Parrish, goes into depth on creating the perfect balance between simplicity, beauty, and features. If you dream about being the one on stage at the Apple Design Award one day, be sure to read Chapter 7. Chris must have eaten his own dog food as his app for sending digital postcards, Postage, won the award in 2009.

Keith Peters comes from the Flash world and delivers, in Chapter 8, concrete solutions to real-world challenges when porting games that were designed for the big screen of a desktop computer to our favorite device with no keyboard and a small, touch-sensitive screen. You'll realize that essential code doesn't have to be complicated.

If you're even the least interested in typography, you'll want to jump directly to Jürgen Siebert's Chapter 9 about FontShuffle, the application that lets you browse through 500 years of type design. You'll learn about the anatomy of letters and gain the knowledge to choose the right fonts for your programming projects and how to get maximum readability on screen or paper.

In Chapter 10, you'll see another example of how not having a programming background can pave the way for creating beautiful and well-designed iPhone apps that are so much more joyful to use than their large-screen Web counterparts. Snow Reports is one such app, and it's hard to believe Eddie Wilson first had to climb the Objective-C learning curve he found steeper than the slopes his app is reporting from.

There you have it: an array of interesting user interface projects lined up for you. Go ahead and dig in.

David Barnard

Company: ***App Cubby***

Location: ***San Marcos, TX***

Former Life As a Developer: ***Recording engineer/producer***

I don't actually have any specific skills related to iPhone development. I don't do the programming and learned everything else along the way.

Life as an iPhone Developer:

Gas Cubby – Utilities - Sensible Car Care.

Trip Cubby – Finance – Mileage Log Made Simple

Health Cubby – Health & Fitness – Social Fitness

What's in This Chapter:

My Journey

Learning From Apple

To Tap or Not to Tap

Usability Testing on the Cheap

Fit and Finish

Key Technologies:

- ***Touch Input***
- ***Usability Testing***
- ***Data Entry Models***
- ***UI Design***

Chapter 1

App Cubby

Creating amazing iPhone applications is quite a bit more involved than it would seem at first glance. Though many gimmicky applications have made money in the App Store, building an elegant, easy-to-use application that solves a real problem or provides meaningful entertainment is the best way to guarantee long-term success. This chapter explores my journey in founding an iPhone development company and building several well-respected, profitable applications.

From Fanboy to Developer

I came to the iPhone platform not as an experienced developer, seasoned entrepreneur, or even programming hobbyist but as a rabid fan. I happened to be traveling in China in January of 2007 and vividly remember sitting in a Beijing hotel lobby, paying way too much for subpar Internet access and trying desperately to get news on the Macworld keynote. Did Apple actually announce a phone? What does it look like? Is it a real Apple device, not like the terrible ROKR I bought and quickly returned?

Fast-forward six months. I'd been watching and rewatching that keynote, reading every blog post, and listening to every podcast. I couldn't wait to purchase an iPhone. My brother called me up late in the afternoon on June 28 and asks me to meet him at the Apple Store. We waited almost 24 hours in line, and I ended up being the first person in San Antonio, Texas to purchase an iPhone. After just a few minutes using the device, it became quite apparent to me that Apple had delivered on the promise of revolutionizing the mobile phone. Being an intellectually curious person, I started thinking quite a bit about this little touch screen device and what made it so compelling. Why was I grabbing it instead of my laptop for certain tasks? How in the world did my Baby Boomer father take to it like a duck to water when he had struggled for years with computers?

That curiosity and the successes of burgeoning jailbreak development community got me thinking about what I might want to create if I were to develop an application for the iPhone. Web applications for the iPhone were functional, but they lacked the power and finesse of Apple's native applications. Rumors started floating around that Apple might actually be announcing an official SDK for native iPhone application development. My

casual ideas about iPhone development slowly started forming into real thoughts of starting a business. I had just gotten married and was quickly realizing that my career as a recording engineer, working late into the night for weeks on end, wasn't congruent with my desire to start a family.

In March 2008, Steve Jobs took the stage and laid out a very compelling opportunity. For a very small fee, anyone could start developing iPhone applications and soon sell those applications to a rapidly growing install base. I was sold. My last scheduled project had just wrapped up in the recording studio, so there was no better time to jump head first into iPhone development.

Having spent the better part of a year casually studying the iPhone and thinking about potential applications, I knew that I would need to start working on this full time if I were going to build anything polished enough to match Apple's default applications. It didn't take much to convince my father and uncle, who are partners in a small business, to help me finance this new venture.

With bootstrap-level funding in the bank and my schedule completely open, I dove into the iPhone SDK. Because I have very little programming experience, the coding was challenging, but I was making progress. After about a week, I had a fully functional, but unpolished, tip calculator! As it turns out, a few tip calculators have made lots of money in the App Store, but at that point, I didn't want to risk my family's money on what I perceived to be a trinket application. My plan was to build a series of highly functional data management applications under the App Cubby brand, starting with a business mileage log called Trip Cubby. I quickly realized that to do this I would need an experienced coder.

Contracting out the coding turned out to be one of the best decisions I made in building App Cubby. Doing so freed me up to focus on the business and, more importantly, designing the applications without having to worry about the code. My informal curiosity about the stellar user experience of the iPhone turned into a systematic study. The multitouch interface we now take for granted was a fundamental shift in the way users interact with computers. Designing an application to fully leverage this amazing new technology was not something I took lightly.

Learning from Apple

As I started thinking about the user interface (UI) for the App Cubby applications and reading various books about UI design, I developed this grand vision of revolutionizing the touch screen interface—until I realized that the iPhone had already done just that. Apple has some of the best UI engineers in the world. Studying how Apple solved various UI challenges in their own applications is the absolute best place for any iPhone developer to start planning a UI.

In an attempt to create a distinct look, many iPhone developers consciously choose to ignore the UI conventions established by Apple in the iPhone's default applications (Phone, Messages Mail, Maps, Photos, etc.). There are definitely some interesting and innovative UI implementations that don't conform to Apple's UI conventions, but for

most developers, sticking to Apple's published *iPhone Human Interface Guidelines* will take you a long way in making a more user friendly application. Speaking of the *iPhone Human Interface Guidelines*, I think that every iPhone developer should read that document cover to cover (multiple times even). My copy is the digital equivalent of a well-worn book, complete with highlights and notes all over the PDF.

After a couple times reading through the *iPhone Human Interface Guidelines*, it really sank in that the default applications created by Apple are some of the most frequently used on the iPhone and therefore the most familiar to the average iPhone user. Breaking from Apple's UI conventions forces the user to relearn certain actions and creates a bit of cognitive dissonance as the user switches among various applications with contradictory UI implementation. Apple's applications are not completely consistent, but they do contain certain patterns and methodologies that, if implemented, make it easier and more natural for users to quickly and effectively grasp the functionality of any application.

Let's look at data entry for a minute. Since I was planning to build a series of data management applications, I spent quite a bit of time studying how Apple addressed the challenge of entering lots of data into an iPhone application. The Calendar application provides a great example of Apple's hierarchical data entry paradigm.

The first level is what I'll call the main view (see Figure 1-1). This view aggregates multiple entries into a bird's eye view and allows the user to quickly scan a lot of information and easily find the information of interest.

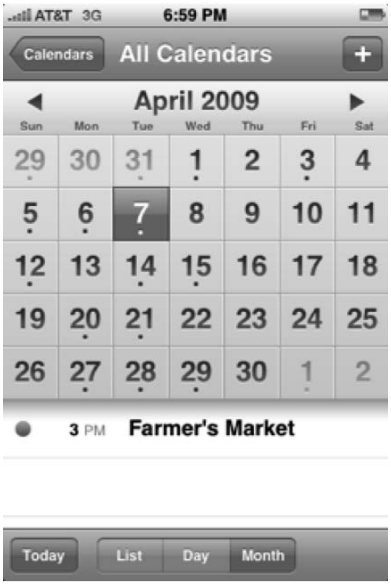


Figure 1-1. *The main view*

Tapping on a summary row (i.e., the *Farmer's Market* row in Figure 1-1) in the main view takes you to the detail view (see Figure 1-2). Here, all the relevant data is displayed in detail.

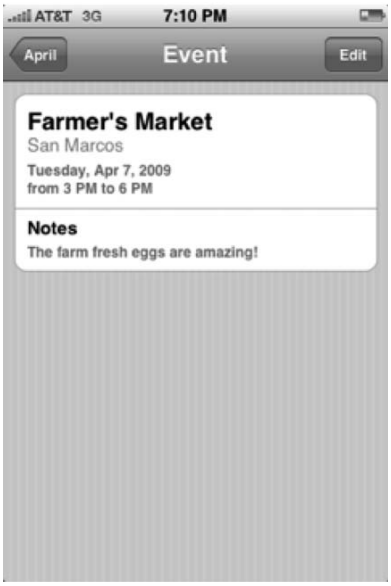


Figure 1-2. *The detail view*

Tapping the *Edit* button in the top-right corner (or creating a new record from the main view) takes you to the data entry overview in the *Edit* screen, shown in Figure 1-3.

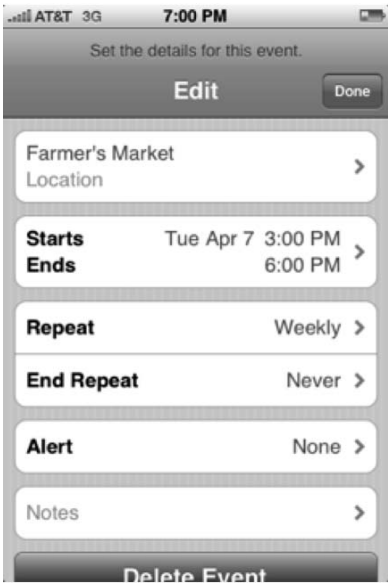


Figure 1-3. *The data entry overview*

In Figure 1-3, all available fields are presented in a grouped table view. Tapping a field takes you to a data entry view, which is shown in Figure 1-4.

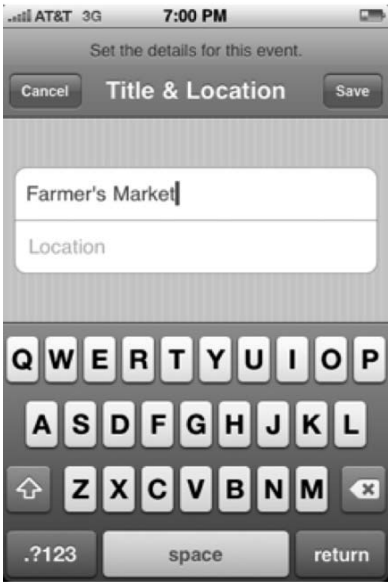


Figure 1-4. *The data entry view*

This is where the magic happens: a keyboard, keypad, picker, or list pops up, and the user actually enters data.

In an attempt to streamline data entry, I've seen a lot of developers skip the view shown in Figure 1-4 in favor of allowing the keyboard to pop up over a long list of fields. Figure 1-5 shows what that shortcut might look like in Gas Cubby.

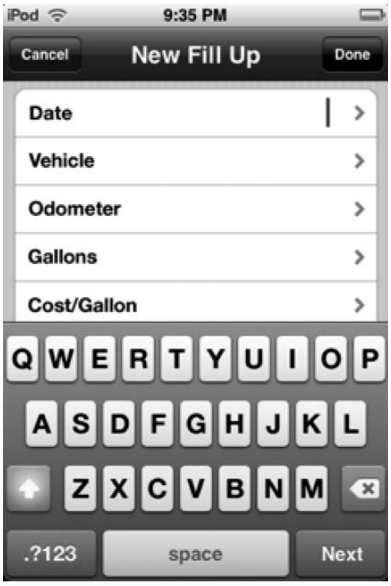


Figure 1-5. *An alternative, "streamlined" data entry screen*

At first glance, this shortcut might seem more efficient than Apple's approach, but in my experience, it causes quite a few accidental taps. While attempting to scroll, users can quite easily accidentally activate a field or tap a button. Combining the data entry overview and the data entry view in a single view takes focus away from the task of actually entering data and makes the user cognizant of the need to tap and swipe carefully.

Focus is the beauty of Apple's data entry paradigm. Because the keyboard takes up so much space on the screen, it's best to have data entry focused on a single field or small group of fields that fit above the keyboard.

Even the Mail application (see Figure 1-6) doesn't hide data entry fields. The Mail data entry view does scroll, and you can access additional data entry fields by tapping on the *Cc/Bcc, From* row. Even so, when you first create a new mail message, every immediately editable field is contained within the view. None are hidden behind the keyboard.

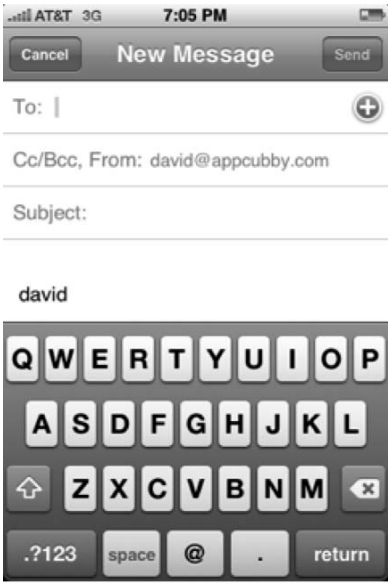


Figure 1-6. *The data entry view in Mail*

Figures 1-7 through 1-10 show how I implemented what I learned about data entry into Gas Cubby.

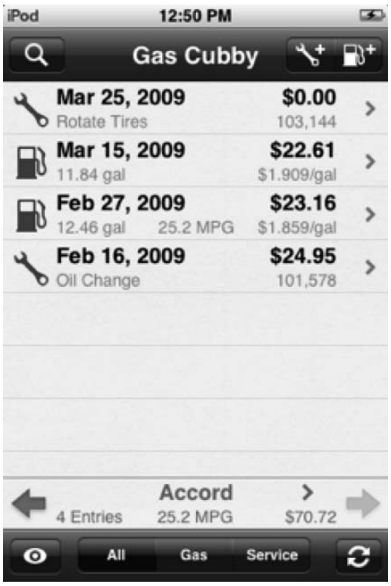


Figure 1-7. *The Gas Cubby main view*

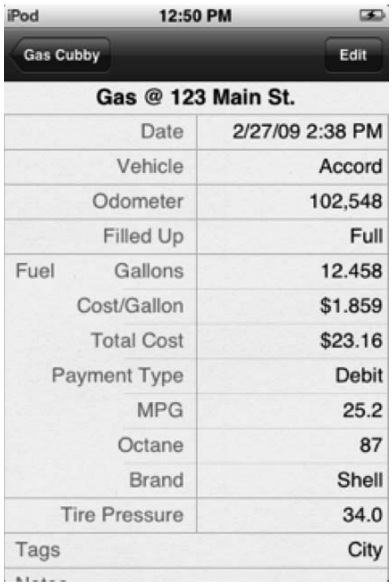


Figure 1-8. *The Gas Cubby detail view*

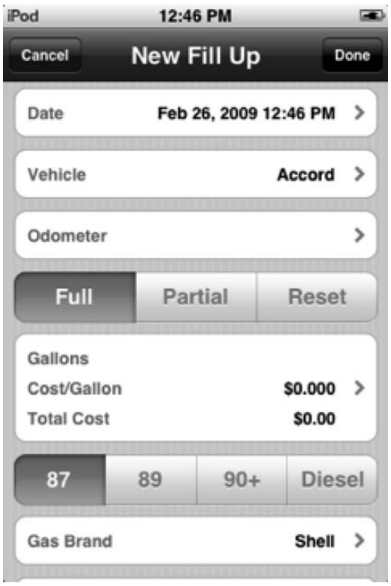


Figure 1-9. *The Gas Cubby data entry overview*

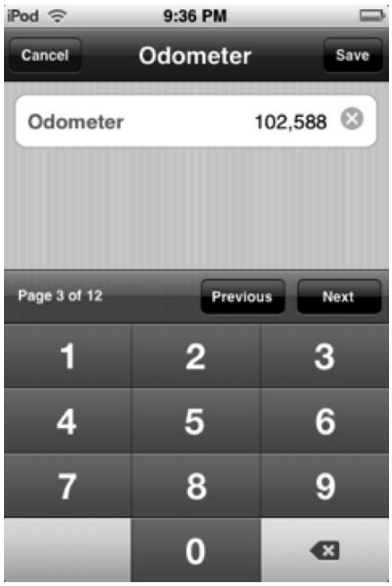


Figure 1-10. *The Gas Cubby data entry view*

The UI is distinct but also uses the Apple UI conventions any iPhone user will find easy to use.

You may notice that the Gas Cubby data entry view (see Figure 1-10) has an extra toolbar not used in the Calendar application. At times, a user might want to quickly enter data into all available data fields. In this case, navigating back and forth to the data entry

overview (see Figure 1-9) wastes time. Rather than breaking from Apple's conventions, I decided to mesh the standard hierarchical data entry model with the shortcut toolbar from Safari, shown in Figure 1-11. This toolbar allows users to quickly fill in every available field using the Previous and Next buttons or use the hierarchical navigation to more efficiently enter data only in certain fields.

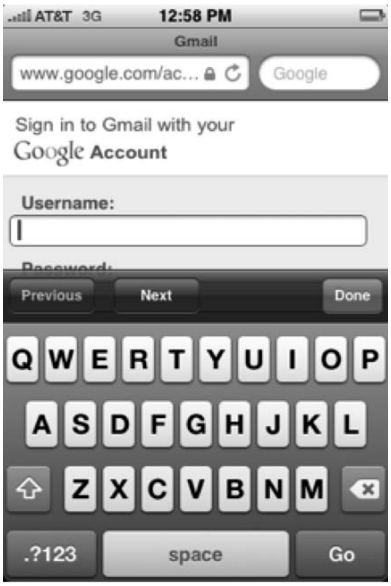


Figure 1-11. *The data entry shortcut toolbar in Safari*

Exploring Apple's data entry paradigm and implementing my own version challenged me to think deeply about why Apple chose this particular implementation and what designs may have been left on the cutting room floor. Some people complain about the perceived inefficiency of Apple's data entry paradigm, but for most users an intuitive interface actually outperforms a potentially faster, but more ambiguous interface. Efficiency is defined as "achieving maximum productivity with minimum wasted effort or expense." There are lots of ways for an iPhone UI to waste user effort, but wasting taps seems to be the focus of most left-brained iPhone developers.

To Tap or Not to Tap?

When I first started mocking up UI elements for Trip Cubby, every feature was measured by the number of taps required to accomplish a specific result. My thought was that by minimizing the number of taps, I was creating a more efficient application. But as I continued studying design and started actually using the early builds of Trip Cubby, my initial ideas and assumptions about what makes a truly efficient UI on the iPhone were thrown out the window.

One of the keys to creating great UI on the iPhone is taking a step back and thinking a bit about how users actually interact with the iPhone—with their fingers. Yes, that's incredibly obvious, but something so obvious generally caries significance that few people take the time to explore.

The finger is an incredibly efficient pointing device, far more efficient than the mouse. When the mouse was first introduced, it revolutionized human interaction with computers. I would argue that Apple's multitouch interface will, in time, prove to be even more revolutionary.

A mouse manifests an unnatural disconnect between the motion of the user's hand and the action on screen. Most people these days have used a computer enough to be somewhat accustomed to the mouse, but if you watch someone use a computer for the first time, you'll see a definite learning curve to using a mouse! I've even noticed a bit of a learning curve when using a new mouse or tracking settings with which I'm unfamiliar.

But the human finger, that's something just about every human being in the world is quite accustomed to using. We've learned since birth how to control our fingers with an amazing level of precision and speed. Imagine if you were to attempt playing the piano with a mouse. Your tune would be choppy and unmusical at best. A touch-screen piano, however, would be playable, even if it didn't match a real piano for feel and accuracy.

The more I thought about the finger as a means of interaction the more intrigued I was by how drastic the shift was from the mouse to multitouch. That's when it finally hit me. Taps are cheap!

If the appropriate action is obvious to the user, the time actually required for that user to tap the proper spot on the screen is miniscule. Confusion about where to tap wastes far more time than an extra tap.

Again, this conclusion may seem quite obvious. After all, ambiguity has been a challenge in all human computer interfaces, and reducing ambiguity has been one of the pillars of good interface design. But the iPhone is the first graphical computer interface where the speed and precision of the pointing device makes the physical action of pointing almost irrelevant when considering the time it takes to accomplish a specific result. Let that sink in for a minute—taps are cheap.

Understanding the finger as an input device is key in the process of making good interface decisions for the iPhone.

Here's quick example: Let's say you are designing a simple yes/no questionnaire. Each participant will be asked a series of questions verbally and should respond by selecting "yes" or "no." Some will be responding using a computer and mouse; others will be responding using an iPhone. To make the interface as intuitive as possible, you quickly decide that the *Yes* button should be green and the *No* button red (assuming a US audience and ignoring for a moment other cultural interpretations of color). Both buttons should have very easy-to-read text that is visually isolated from the background. Figure 1-12 shows what these buttons might look like on the iPhone (if you didn't hire a graphic artist).

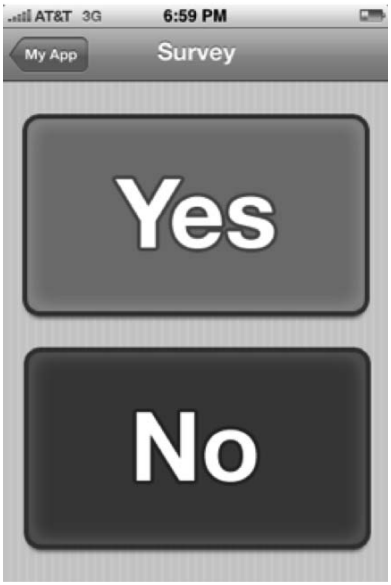


Figure 1-12. *A sample survey application*

Since the buttons have already been created, it's easy enough to just use the same buttons on the computer. Now, you start administering the survey and observe the action and thought required for users to select an answer.

On the computer, the user first must decide what button corresponds to the correct answer (identification). With clearly labeled, color-coded buttons, the process of identification is quite quick. Next, the user must navigate the mouse to the appropriate position on the screen (positioning). Depending on the size of the button, the time it takes to visually locate the mouse pointer on the screen, the tracking precision and speed of the mouse, the position of the mouse on the tracking surface, and even the experience of the user, getting the mouse to the appropriate position on the screen can actually take significantly longer than the process of identification. Finally, the user has to click the mouse button (action). With a mouse, clicking is generally instinctual, but inexperienced computer users may still need a split second to decide whether to right-click or left-click.

On the iPhone, the user decides which button corresponds to the correct answer (identification) and then moves a finger into contact with the appropriate button on the screen (positioning and action).

In this scenario, the buttons on the iPhone and computer are exactly the same and should both be very easy to properly identify. Positioning and action are where the rubber meets the road. I haven't actually done this test and don't have any hard data, but it should be pretty obvious that moving from a mouse to a finger as a pointing device is a fundamental shift in the use of graphical interfaces.

At this point, some of you may be thinking that this was an unfair comparison, as the keyboard would obviously have been the best way for the computer user to interact with this survey by pressing Y for “yes” and N for “no.” That’s just the kind of logic-driven, computer-geek thinking that must be put aside when designing UI for the iPhone. Shortcuts are nice, but the average computer user can barely remember the keyboard shortcuts for copy and paste, much less the long list of shortcuts available for most modern applications.

To use the keyboard in the survey scenario, the users would first have to be told to press Y for “yes” and N for “no.” Each time a question was asked, they would have to make the proper decision and then think “press Y for ‘yes’ and N for ‘no’” before pressing the proper key. Eventually, the user would be accustomed to the interaction and become more efficient. But that’s why the mouse was so revolutionary in its time and why the multitouch interface is an even bigger leap: each smoothes out the learning curve and mitigates steps from thought to outcome.

A physical keyboard can be quite efficient in the positioning and action phase (if the user is a touch typist), but the time it takes to identify the proper action is where the slowdown occurs. The user must first learn the commands and then remember and properly execute them. The mouse and graphical interface have generally made improvements in the speed of identification; it’s easier to identify the symbol or actual name of an action than it is to remember a key command for that action, but this improvement in identification came at the expense of having to position the cursor before taking action.

The multitouch interface has the potential to combine the strengths of both traditional forms of user input. As a graphical interface, iPhone UI can use names and symbols to help users quickly identify the appropriate action. As a touch screen, the iPhone benefits from the speed and intuitive nature of using a finger as an input device.

The potential benefits of the multitouch interface are, however, easily squandered with a bad UI. That’s where counting taps, which I would imagine is the gut instinct of most developers coming from the world of costly mouse clicks, can be dangerous. If actions are ambiguous, the physical speed of tapping is completely negated by the time it takes the user to identify the proper action. Using a confusing interface to save a few taps is the iPhone equivalent of keyboard shortcuts. Sure, some users will read your 50-page manual and end up a bit more efficient, but the learning curve is steep, and average user will continue to struggle.

Creating a user-friendly iPhone application is about striking a balance between efficiency of identification and efficiency of tapping. Because tapping is quick and intuitive, finding that balance generally leans in the direction of additional taps. There are times when counting taps might help make a great UI a bit more efficient, but starting the design process by counting taps will generally lead you down the wrong path. When designing for the iPhone, ambiguity is far more costly than taps.

Admitting that taps are cheap doesn’t help UI design unless you also have a good grasp on ambiguity. Once you’ve spent much time on a project, the UI becomes so obvious to you it’s hard to be objective about the ambiguous aspects of your application. The best

way for a developer to get objective feedback about a particular UI implementation is to do usability testing.

Usability Testing on the Cheap

Since most iPhone applications are created by individuals or small teams on tight budgets, I would guess that very few have been subjected to usability testing. And it shows! Just because you don't have a budget or a formal understanding of the theories of usability testing doesn't mean you shouldn't test your application. Usability testing can become very time consuming and expensive if done on a large scale in a traditional way, but there are cheap and free ways to do informal usability testing that will greatly improve any UI.

Finding Users

First, let me clear up the most common misconception: usability testing is not the same thing as beta testing. If you don't test and set a direction for the UI first, beta testers will generally send you into a death spiral of adding unnecessary features and buttons to an interface that has no focus. Beta testers are typically more advanced iPhone users who don't mind taking a few minutes to learn the ins and outs of an application. They will quickly see beyond any usability problems and start brain storming new features and advanced options.

To properly test the usability of an iPhone application, you need to find several "average" iPhone users. I enlisted the help of family and friends, but you might want to put out an ad on craigslist and offer free lunch or an iTunes gift certificate. You're looking for people who know how to use the iPhone but aren't power users. The best way to do usability testing is in person, but it can be done via video conferencing or by having the user just video tape the testing.

Testing Done Right

If you're serious about usability testing, I recommend reading up on the theories and techniques; it's a fascinating field of study. In case you can't be bothered with doing additional research, let me give you a very quick overview to help point you in the right direction.

The first thing to do is to properly condition your test subjects. They need to understand that there are no right and wrong answers. Making mistakes is expected and is actually helpful. Explain that by making a mistake they will actually be helping you find flaws in your application. Don't explain how the application works or give any kind of tutorial on how to use your application. Usability testing is intended to simulate the real-world use of an application, and contrary to the hope of all software developers, most users will never read the manual. You can, however, give a basic description of the intent of your application. In my case, I would say something like, "This is Trip Cubby; it's an application designed to help track and report mileage for taxes or reimbursement."