iPhone Design Award-Winning Projects

Chris Dannen

Apress®

iPhone Design Award-Winning Projects

ISBN-13 (pbk): 978-1-4302-7235-9

ISBN-13 (electronic): 978-1-4302-7234-2

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

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.

Contents at a Glance

Contents

About the Authors

Chris Dannen is a writer specializing in technology and innovation. He writes primarily for *FastCompany* magazine and Bnet.com, where he covers software, mobile technology, Web, mapping and all things Apple. He is also co-author of *Google Voice For Dummies* (Wiley, 2009). Chris received his BA from the University of Virginia and lives in the Williamsburg section of Brooklyn, NY.

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*, *iPhone User Interface Design Projects*, and now *iPhone Design Award-Winning Projects*.

Acknowledgments

A great deal of gratitude is due to all the developers who sat for interviews in order to contribute to this book. Their patience, expertise and colorful explanations make this book what it is.

Special thanks to the developers who answered my incessant questions with extra detail, and took the time to explain ancillary concepts, code and hypotheticals at great length: Jonathan Wegener, Loren Brichter, Brandon Walkin and AJ of Marketcircle; Dave Witonsky for his enthusiasm and Marco Arment for one giant cup of green tea.

Thanks also to Clay Andres for his perennial guidance, and Kelly Moritz, Douglas Pundick, Tracy Brown Collins, and Joachim Bondo for their hard work and careful consideration. Many thanks also to Austin Carr, whose quick transcriptions saved me from the woes of carpal tunnel.

Finally, thanks to my editors at *FastCompany*, Noah Robischon and Lynne d Johnson particularly, for accommodating the crazed schedule I adopted while writing this book.

Introduction

Readme

This is a book about building apps with good design, parsimonious code and aesthetic appeal on the Apple iPhone and iPod Touch. At times, this book references code and give sample projects; other times, it delves into interaction and visual design. It doesn't take a CS doctorate to appreciate or even understand, but some familiarity with programming is assumed.

Who is this book for?

Lots of people. Mac developers looking to do a killer iPhone app might want to know the hard-won lessons or philosophies of others; iPhone hackers looking to step up their game might be trying to figure out how much to emulate Apple. Anyone who appreciates a success story will hopefully enjoy this book, but it's also inspirational.

How technical is it?

Parts of this book assumes basic knowledge of C or other object-oriented programming languages. In some cases, we reference specific Apple frameworks and interface guidelines. It will also help to be familiar with the Apple environment, from the quirks of the App Store to technologies inside OS X.

How were these interviewees chosen?

The five core chapters of this book are based on a series of interviews with five of the 2009 winners of Apple Design Awards (ADAs) for the iPhone. MLB.com, the makers of the sixth ADA winner, MLB.com at Bat, did not consent to be interviewed for this book in order to protect their intellectual property.

The remaining six developers interviewed for this book were chosen by its writer and editors because they buttressed some of the discussions in the core chapters, or because they provided a contrasting philosophy to another developer in the book. It was important not only that these six apps use Apple frameworks competently, but that they also smartly navigated the App Store, its frenetic marketplace, and its fickle economy.

Speaking of money: this book also aims to address the philosophies behind monetizing (or not monetizing) an app. How do I set an optimal price? Should my app be free? How about the upgrade? What are the risks of in-app purchasing? It's not as much fun as talking about code or UI, but sometimes monetization means the difference between designing the app you want, or compromising to cover your overhead.

Lastly, this book tries to garner thoughts and opinions from developers building a wide variety of software. Several of the apps highlighted herein are tied into desktop companion apps or websites; others discuss the construction and revision of sequels. Some of the developers themselves aim to startup iPhone-only shops, while some are veteran Mac developers or new-to-Apple developers looking to experiment. But they’re all motivated by a single question: how do I build a better iPhone app in concept and in practice?

Chris Dannen

Part I

Innovating Beyond Apple's Design Standards, While Maintaining Apple's Logic for Consistency, Clarity, and Usability

With over 100,000 apps in the App Store, how have Facebook and Tweetie managed to rise above the rabble? Used by millions of iPhone owners and accessed just as often as many of Apple's native apps, both these social network clients wield incredible influence, not only over their users, but also over the way UI standards come to be accepted by the iPhone masses in general. These are no normal apps. They are apps that help make the platform.

Tweetie and Facebook are incredibly deep and beautifully-engineered pieces of software. The two are made for very different use--Tweetie is borne of singular purpose, while Facebook is evolving to be a Swiss Army app--but they are alike in one particular trait: personality. Both developers have crafted their vision of iPhone UI with stubborn confidence, but also even-handedness. Where they believe Apple has shown ingenuity, they pay competent homage. But where they believe Apple has lapsed, they gladly pick up the pace of innovation and carry it forward.

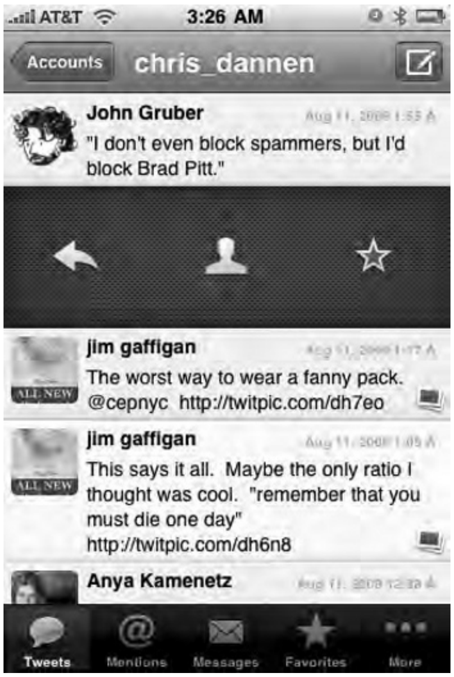
This is no small assumption on an Apple device, which, like other Apple products, has come to vaunt its remarkable ease of use and its long, careful development as the reasons for its success. Loren Brichter and Joe Hewitt, the developers featured in this section, don't seem worried. It's a combination of humility and precociousness that could only be called Jobsian.

Chapter 1

Tweetie

Developer name: Loren Brichter
Development Company: Atebits
Tags: Layout; Efficient Code; Workflow
URL: `http://atebits.com`

"I can't go into details because I think everything is under NDA for, like, the next 20 years," says Loren Brichter, the founder and sole developer at Philadelphia-based Mac shop atebits, and developer of Tweetie (Figure 1–1). He's talking about the top-secret program he worked on at Apple: the iPhone.



John Gruber
"I don't even block spammers, but I'd block Brad Pitt."

jim gaffigan
The worst way to wear a fanny pack. @cepnyc http://twitpic.com/dh7eo

jim gaffigan
This says it all. Maybe the only ratio I thought was cool. "remember that you must die one day" http://twitpic.com/dh6n8

Anya Kamenetz

Figure 1–1. *Tweetie's innovative UI won it a cult following.*

He stayed for a year—the most exciting year of his life, by his own account. But having grown up on Manhattan's Upper West Side and gone to Tufts University in Boston, Brichter was a Right Coast kid and he couldn't stay away. He moved back East, sat down, and did the only thing he knew how to do: he wrote a Mac app.

"My first product was a little drawing app called Scribbles," he says. "It kept me a live, it made a little bit of money, but it wasn't hugely successful." He was good with Cocoa and OpenGL; the projects that had gotten him hired at Apple were an iTunes visualizer and a soft-body physics simulator. Once assigned to the iPhone project, he had worked in the guts of Cocoa, at the UIKit level and below. He was a programmer's programmer. So it was no surprise that when he first signed up for Twitter in November 2007, he was more or less disgusted by the brevity of 140-character tweets. "I signed up, used it for five minutes, and decided: this is stupid," he says.

It would be a year before he started using it again, to keep up with the tweets of tech pundit John Gruber. Why? "Gruber is interesting guy," Brichter says. He found there were actually lots of other interesting guys on Twitter. "People joke about Twitter, that it's stupid and superficial," Brichter says. "And if you follow superficial people, it's superficial. But if you follow interesting people, it's interesting."

A Billion Bad Twitter Apps

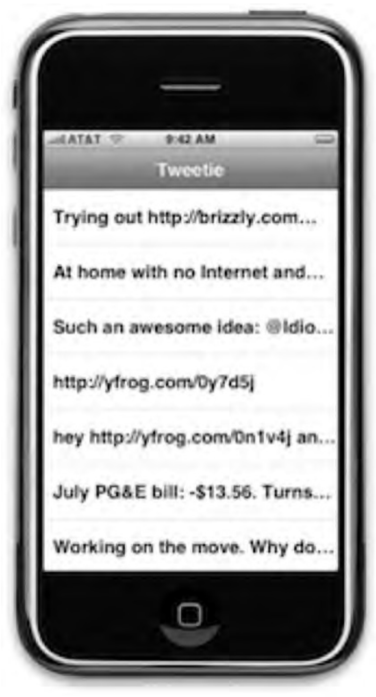
It was around the same time that his Verizon Wireless contract expired and he finally got an iPhone. He started scouring the App Store. "I realized there are no good Twitter apps," he recalls. "But there are a billion bad ones." He figured he could probably write a better app. "What triggered me to do it? I was playing with Twitterific, which I used, and everybody used. I thought: I wonder why the scrolling is so slow? I wonder if I can make it faster." In an hour, he had built a prototype of a list of fast-scrolling tweets. Then, after a two-week paroxysm of coding, he had built Tweetie, shown in Figure 1–1, which is as of this writing the most popular mobile Twitter client on any platform, and the most popular Twitter app for iPhone.

Brichter attributes Tweetie's meteoric rise in popularity to a cocktail of luck, quality programming, and some well-timed publicity. But what really lives at the core of his success—and Tweetie's—is an absolute intolerance for mediocrity. "I have a shit-list of Twitter apps that just drive me insane," he says. "Apple lays out all these guidelines and conventions about how to write an iPhone app, and these developers basically looked at them and did the exact opposite in every single case."

He's not looking for perfection, however—just parsimony. If it's one concept of design and interaction that gets Brichter excited, it's economy of energy, be it in actual usage scenarios or in software development. Ask him what aggravated him about the existing selection of Twitter apps back in November of 2008, when he wrote Tweetie, and he doesn't talk about their ugliness or their complexity. Even the slowness that bugged him in Twitterific is merely a symptom of flawed thinking. "The problem isn't with how the other apps used Twitter's API, it's with the way they interacted with the iPhone OS," he says. "Either they were doing something completely custom, or completely wrong." His antipathy wasn't even aimed at Twitterific, though it was the immediate catalyst for

Tweetie. “To tell you the truth, I didn’t have a lot of beef with Twitterific,” he says. “They were the ADA winner from the year before, and everyone loved the app. It just didn’t jibe with the way I used Twitter.”

It’s not that Brichter is a loyal Apple devotee, either; he has beef with plenty of the native apps. “On the SMS app, the ‘Send’ button is right next to the keyboard,” he quibbles, “so you hit it accidentally. There’s no reason they couldn’t have put it up in the nav bar.” Mail too was a cautionary tale for Tweetie. “Everything in Mail is twice as many taps [as with Tweetie]. You have the account list, then the folder list, and then you get into the message list,” he says. “I want to be able to go into an inbox with one tap, not two or three.” (Figure 1–2 shows the first tweet feed prototype; Figure 1–3 shows a second iteration.)



Trying out http://brizzly.com...

At home with no Internet and...

Such an awesome idea: @Idio...

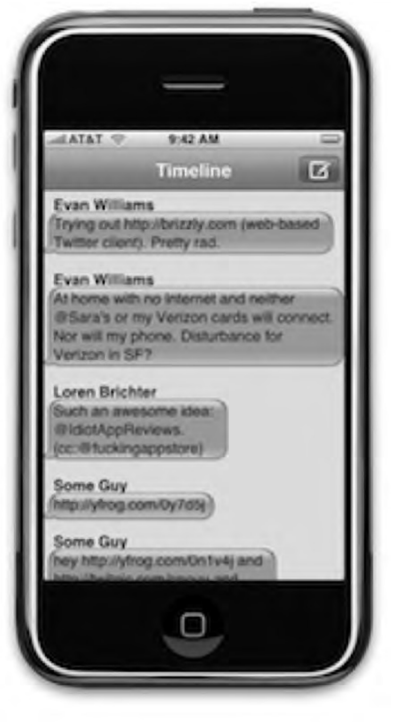
http://yfrog.com/0y7d5j

hey http://yfrog.com/0n1v4j an...

July PG&E bill: -$13.56. Turns...

Working on the move. Why do...

Figure 1–2. *The first Tweetie fast-scrolling prototype.*



Evan Williams
Trying out http://brizzly.com (web-based Twitter client). Pretty rad.

Evan Williams
At home with no internet and neither @Sara's or my Verizon cards will connect. Nor will my phone. Disturbance for Verizon in SF?

Loren Brichter
Such an awesome idea: @IdiotAppReviews. (cc: @fuckingappstore)

Some Guy
http://yfrog.com/0y7d5j

Some Guy
hey http://yfrog.com/0n1v4j and

Figure 1–3. *Another scrolling iteration, this one more iChat-inspired.*

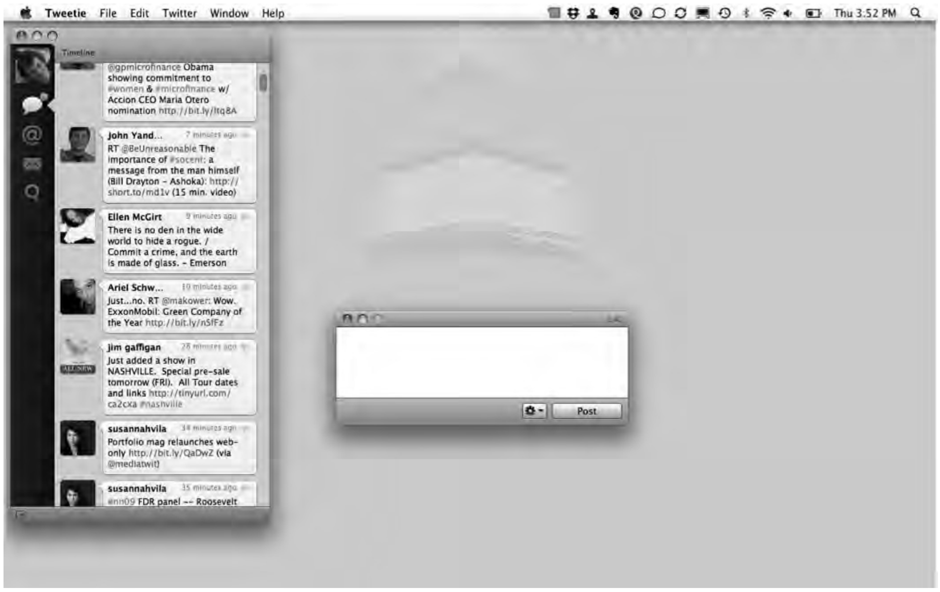
The atebits way of using something, distilled, is the pursuit the path of least resistance. "I got lucky, because the way I originally wrote Tweetie made sense in terms of how you should interact with a Twitter app," he says. His idea of how a user "should" use Twitter is based the navigation Apple Mail, made better: each tap in a tweet's menu pushes you rightward into a deeper folder. That's how Tweetie manages to pack in Recent Tweets, Search @, Following, Followers, and Block/Unblock without a mess of buttons or oddball menus. "It doesn't seem complicated—that's how every Twitter app should work," he says. "But yet that's not how Twitter apps work."

The most popular functions aren't made available through more buttons, or more menus—just swipe, as if you're deleting an email or a text, and you have the ability to star, @reply, or see the profile of the user whose tweet you're investigating (though you can also do this by clicking a tweet, if you're not aware of the shortcut). "My philosophy was: don't fight the framework," he says. "UI Kit APIs are so insanely beautiful that you can pick them up super-fast, even if you don't know how to program. It provides all this awesome functionality, but all these other apps do things their own custom way. But if you just embrace the UI Kit philosophy, you can build an app like Tweetie really fast." (Figure 1–1 shows the swipe shortcut.)

Easy for the ex-Apple iPhone wonk to say, right? Well, sort of. "At Apple I didn't do any app stuff except some performance tuning for other teams—making apps faster, doing some fancy looking transitions. I didn't really have any app building experience at all, so when I sat down to start writing Tweetie, I was learning the process from scratch."

The Minimalist Flourish

While Brichter might have a thing for parsimony, the fine-tuning he did during his year at Apple is readily apparent in both Tweetie and Scribbles in a handful of beautiful actions. Make a command—a new document or tweet, or the preferences pane—and the window slips out from under the menu bar with a flourish, in a kind of reverse-genie effect inspired by the dock. Things in both apps move uncannily fast; even on a dual-core Macs where waiting is a rarity, the file browser practically rockets out of a tweet's title bar. Atebits apps are compact, purpose-built and deceptively robust, like race-tuned Mini Coopers of the Cocoa realm. (Figure 1–4 shows a tweet in the making, using Tweetie for Mac.)



@gpmicrofinance Obama showing commitment to #women & #microfinance w/ Accion CEO Maria Otero nomination http://bit.ly/ltq8A

John Yand... 7 minutes ago
RT @BeUnreasonable The importance of #socent: a message from the man himself (Bill Drayton - Ashoka): http://short.to/md1v (15 min. video)

Ellen McGirt 9 minutes ago
There is no den in the wide world to hide a rogue. / Commit a crime, and the earth is made of glass. - Emerson

Ariel Schw... 10 minutes ago
Just...no. RT @makower: Wow. ExxonMobil: Green Company of the Year http://bit.ly/n5fFz

jim gaffigan 26 minutes ago
Just added a show in NASHVILLE. Special pre-sale tomorrow (FRI). All Tour dates and links http://tinyurl.com/ca2cxa #nashville

susannahvila 34 minutes ago
Portfolio mag relaunches web-only http://bit.ly/QaDwZ (via @mediatwit)

susannahvila 35 minutes ago
#nn09 FDR panel -- Roosevelt

Figure 1–4. *Tweetie for Mac, which uses the same codebase as Tweetie 2 for iPhone, contains subtle, rapid animations.*

Economy of design is often accompanied by minimalism, and Brichter's work style is fittingly Spartan. He doesn't use Interface Builder; all of Tweetie versions 1 and 2 were built programmatically in Xcode. And unlike most of the Mac developers in this book, uses just one machine and just one screen. "A few months ago I got a 17-inch MacBook Pro, and my Mac Pro has been in storage ever since," he says. It took Brichter just five days of coding to build the Twitter-iPhone core inside Tweetie 1, a few days doing the user interface, and a few more in the hands of beta testers; the whole app weighed in at 30,000 lines of code.

It is perhaps because of Twitter's straightforwardness that Brichter's Tweetie project makes for such an excellent example for developers. Unlike most apps, Tweetie has the benefit (and the onus) of an extant paradigm: the Web version of Twitter The extent to which Tweetie (or any Twitter iPhone app) succeeds is based largely on one question: how much additional cognitive load will it take me to use Twitter using this app instead of the Web?

To give Tweetie parity with the Web experience, Brichter first built in all the features an average user would want, but he didn't stop there; he lets power-users navigate reply chains, upload pictures to Twitpic.com, view trends, do searches, search nearby users using the phone's A-GPS, and even built a bookmarklet that users can deploy in mobile Safari to send links to Tweetie. In 1.2, the first major feature update of release, he added Instapaper support, landscape keyboard, image compression control for Twitpic uploads, those Mail-like swipe shortcuts, stock quote links, and the ability to forward direct messages to email (for more on Instapaper, see Chapter 13). If you're a Twitter Web user, you know that a whole lot of those features don't even appear anywhere on Twitter itself.

So features he had in spades—but having tons of features is often anathema to usability. Since it was Twitterific's slow scrolling that compelled him to build Tweetie in the first place, Brichter first set out to invent a new way the iPhone rendered its table cells. Most apps—in accordance Apple's own tutorials—force the phone to render a morass of subviews of labels and images, but Brichter's code pre-blends everything together using CoreGraphics. Once it renders that first frame, it hands one opaque static view over to CoreAnimation without the need to rely on the GPU to do any blending Brichter was so convinced that his fast-scrolling code was superior to Apple's that he posted a tutorial on his web site, where he makes his case: "If you think about what is happening in terms of overdraw, having one big view per table cell is a big win, because CoreAnimation will only touch a single given pixel on the screen once rather than multiple times (potentially, depending on how much overlap your old view hierarchy had)," he wrote.[1]

As with Tweetie, Brichter's sample project relies on just one class of object for everything: ABTableViewCell.h and ABTableViewCell.m. In Listing 1–1, he creates a sample list of words with a first- and last-name field in two separate fonts as in the Contacts app.

List 1–1. *Creating a first- and last-name field*

```
//
//  FirstLastExampleTableViewCell.m
//  FastScrolling
//
//  Created by Loren Brichter on 12/9/08.
//  Copyright 2008 atebits. All rights reserved.
//

#import "FirstLastExampleTableViewCell.h"
```

[1] http://blog.atebits.com/2008/12/fast-scrolling-in-tweetie-with-uitableview/

```
@implementation FirstLastExampleTableViewCell

@synthesize firstText;
@synthesize lastText;

static UIFont *firstTextFont = nil;
static UIFont *lastTextFont = nil;

+ (void)initialize
{
        if(self == [FirstLastExampleTableViewCell class])
        {
                firstTextFont = [[UIFont systemFontOfSize:20] retain];
                lastTextFont = [[UIFont boldSystemFontOfSize:20] retain];
                // this is a good spot to load any graphics you might be drawing in -
drawContentView:
                // just load them and retain them here (ONLY if they're small enough
that you don't care about them wasting memory)
                // the idea is to do as LITTLE work (e.g. allocations) in -
drawContentView: as possible
        }
}

- (void)dealloc
{
        [firstText release];
        [lastText release];
    [super dealloc];
}

// the reason I don't synthesize setters for 'firstText' and 'lastText' is because I
need to
// call -setNeedsDisplay when they change

- (void)setFirstText:(NSString *)s
{
        [firstText release];
        firstText = [s copy];
        [self setNeedsDisplay];
}

- (void)setLastText:(NSString *)s
{
        [lastText release];
        lastText = [s copy];
        [self setNeedsDisplay];
}

- (void)drawContentView:(CGRect)r
{
        CGContextRef context = UIGraphicsGetCurrentContext();

        UIColor *backgroundColor = [UIColor whiteColor];
        UIColor *textColor = [UIColor blackColor];

        if(self.selected)
        {
                backgroundColor = [UIColor clearColor];
                textColor = [UIColor whiteColor];
        }

        [backgroundColor set];
        CGContextFillRect(context, r);

        CGPoint p;
        p.x = 12;
        p.y = 9;

        [textColor set];
        CGSize s = [firstText drawAtPoint:p withFont:firstTextFont];

        p.x += s.width + 6; // space between words
        [lastText drawAtPoint:p withFont:lastTextFont];
}

@end
```

ABTableViewCell.m reads:

```
#import "ABTableViewCell.h"

@interface ABTableViewCellView : UIView
@end

@implementation ABTableViewCellView

- (void)drawRect:(CGRect)r
{
        [(ABTableViewCell *)[self superview] drawContentView:r];
}

@end



@implementation ABTableViewCell
```

ABTableViewCell.h reads:

```
#import <UIKit/UIKit.h>

// to use: subclass ABTableViewCell and implement -drawContentView:

@interface ABTableViewCell : UITableViewCell
{
        UIView *contentView;
}

- (void)drawContentView:(CGRect)r; // subclasses should implement

@end
```

RootConroller.m reads:

```
//
//  RootViewController.m
//  FastScrolling
//
//  Created by Loren Brichter on 12/9/08.
//  Copyright atebits 2008. All rights reserved.
//

#import "RootViewController.h"
#import "FastScrollingAppDelegate.h"
#import "FirstLastExampleTableViewCell.h"

@implementation RootViewController

- (void)viewDidLoad
{
        self.title = @"Fast Scrolling Example";
    [super viewDidLoad];
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return 100;
}

static NSString *randomWords[] = {
@"Hello",
@"World",
@"Some",
@"Random",
@"Words",
@"Blarg",
@"Poop",
@"Something",
@"Zoom zoom",
@"Beeeep",
};

#define N_RANDOM_WORDS (sizeof(randomWords)/sizeof(NSString *))

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
        static NSString *CellIdentifier = @"Cell";

        FirstLastExampleTableViewCell *cell = (FirstLastExampleTableViewCell
*)[tableView dequeueReusableCellWithIdentifier:CellIdentifier];
        if(cell == nil)
        {
                cell = [[[FirstLastExampleTableViewCell alloc]
initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier] autorelease];
        }

        cell.firstText = randomWords[indexPath.row % N_RANDOM_WORDS];
        cell.lastText = randomWords[(indexPath.row+1) % N_RANDOM_WORDS];
```