More iPhone Cool Projects

Cool Developers Reveal the Details of Their Cooler Apps and Discuss Their iPad Development Experiences

Danton Chin
Claus Höfele
Ben Kazez
Saul Mora
Leon Palm
Scott Penberthy
Ben Britten Smith
Chuck Smith
David Smith
Arne de Vries
Joost van de Wijgerd

Apress®

More iPhone Cool Projects: Cool Developers Reveal the Details of Their Cooler Apps and Discuss Their iPad Development Experiences

ISBN-13 (pbk): 978-1-4302-2922-3

ISBN-13 (electronic): 978-1-4302-2923-0

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

Publisher and President: Paul Manning
Lead Editor: Clay Andres
Development Editors: Douglas Pundick, Matthew Moodie, and Brian MacDonald
Technical Reviewer: Ben Britten Smith
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, Matthew Moodie, Duncan Parkes, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Coordinating Editors: Candace English and Debra Kelly
Copy Editor: Katie Stence
Compositor: MacPS, LLC
Indexer: BIM Indexing & Proofreading Services
Artist: April Milne
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 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 `www.springeronline.com`.

For information on translations, please e-mail `rights@apress.com`, or visit `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 `www.apress.com/info/bulksales`.

The source code for this book is available to readers at www.apress.com. You will need to answer questions pertaining to this book in order to successfully download the code.

My gratitude and thanks to Mom and Dad (they would have liked this), to my wife Carol for everything over the years, and to our wonderful and artistically, musically talented son Tim. Also, thanks to Robert and Elizabeth Bergenheim, Elise Falkinburg and Errol Frankel, and their wonderfully intelligent sons, John and James Frankel.

—Danton Chin

Thanks to my wife and daughters, who inspire me daily to help make the world a better place. And for letting me buy that awesome new MacBook Pro (you know which one).

—Saul Mora

To my mother, who taught me to pursue excellence, and to my father, who always inspired creativity in me. Also to Andrea Zemenides, for being cute and little.

—Leon Palm

To the beautiful women in my family: Lisa, Julia, and Taylor. Taylor insists I mention our dog, Jack, and Snickers too.

—Scott Penberthy

To my lovely wife Leonie.

—Ben Britten Smith

To my parents, who were always there for me and gave me the joy of reading. Also to my professor, Dr. Gene Chase, who always brought an incredible amount of enthusiasm to everything he taught, and made computer science inspiring.

—Chuck Smith

To my wife and best friend, Lauren.

—David Smith

Contents at a Glance

Contents

Preface

Dear Reader,

This is the fifth in the Apress series of iPhone Projects books and the first to have the word *iPad* mentioned. To say that we're all hyperaware of the iPad and all that it promises is something of an understatement; will eBooks and Apple's new iBooks store be the killer app for iPad, is this really as "magical" as Steve Jobs says it is, and who are all of these people buying every iPad Apple can manufacture? And yet, *More iPhone Cool Projects* is was written about smaller-screen apps for iPhone and iPod touch. Fear not!

When we started putting this book together, it was still 2009 and the iPad had not been announced. As we were finishing the editorial process, each of the ten chapters was reviewed and updated as appropriate to make mention of iPad considerations. At the same time, we discovered an inherent truth about iPhone and iPad development: all of your iPhone knowledge is invaluable for writing iPad apps, as well. We know this, because each of the 11 contributors (one chapter has coauthors) is moving right into iPad app development.

We urge you think of the lessons learned and code shared in this book as applying to any app you might choose to create using Apple's iPhone OS! In fact, the tools remain the same: Objective-C (with a few exceptions), Cocoa Touch, and XTools. Because of this core of Apple technologies, the best practices also carry across all of the various iPhone OS–running mobile platforms. This is a key theme running through all of the Apress Projects books. Somehow, we had an idea that Apple had more things up its corporate sleeve.

As always, I'd like to mention Dave Mark, our tireless series editor and author of several bestselling Apress titles, including *Beginning iPhone 3 Development*, *More iPhone 3 Development*, and, very soon, *Beginning iPad Development*. In many ways, Dave embodies the positive energy and inspirational spirit that makes the iPhone and iPad developer community such an exciting place to be a part of.

It's in this spirit of collegiality that we have done everything we can to ensure that all of the books in this series are truly useful and enjoyable. We've tried to include something for every style of development, or at least to cover a range of tastes. Please let us know what you think, and we'd be happy to hear about new ideas you may have.

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

Dave Mark
Series Editor, Bestselling Author, and Freelance Apple Pundit

Acknowlegments

What can I say? A book like this cannot exist without the efforts and passion of a great many people. I have read this book. A few times now. It is amazing and I learn new things every time I go through it. It is the product of thousands of combined hours of effort, and I want to give some credit and thanks to the people who made it all possible.

First off, I want to thank all of the authors who took great pains to distill their years of knowledge and experience into words and code for us to learn from, and who patiently took all of our comments, critiques, and requests for yet more code samples but always came back with increasingly better and better material to work with.

I would like to give a shout-out to Clay Andres, the lead editor who originally approached me about this book and basically did all the legwork to gather the authors together and get the project rolling.

I want to give huge thanks to Debra Kelly and Candace English for keeping all of us authors and reviewers herded in the right direction, working on the right things at the right times and keeping the maze of files and revisions and documents in order.

Huge admiration goes out to the development editors Douglas Pundick and Brian MacDonald and our copy editor Katie Stence. They let us authors focus on getting our thoughts onto paper, and they came through and made sure it sounded good and looked professional.

Thanks!

—Ben Britten Smith

Introduction

This is a wonderful book.

I am a working iPhone developer. I spend each and every day of the work week (and most weekends) writing code that is destined to run on someone's iPhone, iPad, or iPod Touch. I have been doing this for a long time now, and yet there is still so much more to learn!

During the course of this book project, I had the task of going through every single chapter, every single line of code, and building every single sample project (often more than once). I don't recall a single chapter that did not provide me with some insight to a problem that I had worked on recently, was currently working on, or am planning to implement in future projects. Some of the stuff I learned I was able to apply immediately to my running projects. I can pretty much guarantee there is something in here for most every developer out there.

Who This Book Is For

This book presupposes that you have some familiarity with iPhone development. Most of the projects presume that you are able to build and deploy apps written in XCode onto your device for testing. If you started with a book like *Beginning iPhone Development* by Dave Mark and Jeff LaMarche (Apress, 2009), then you will be well set to tackle the projects in the following pages.

There are a few chapters that go into some Mac based tools, so it will also be helpful to be familiar with Objective-C and C development with XCode on the desktop. However, if you have used XCode to compile and deploy an iPhone app, then the desktop stuff should be fairly easy to pick up. If you want to learn more, have a look at some books like *Learn C on the Mac* by Dave Mark (Apress, 2009) and *Learn Objective-C for the Mac* by Mark Dalrymple and Scott Knaster (Apress, 2009).

There is even a light dusting of C# in the chapter on Unity3D. What?! C# in an iPhone book? I told you there is something for everyone here. The C# is very simple and mastery is not required to understand the chapter, but if you are interested, check out *Beginning C# 2008* by Christian Gross (Apress, 2007).

Astute iPad developers may notice that all of the sample code and projects in this book are generally built for the iPhone and iPod Touch devices. This is to make sure that we could cover as many devices as possible. All of the concepts and ideas covered here apply equally to the iPad, of course, and all the code runs perfectly well on that device.

What's in the Book

In Chapter 1, Danton Chin delves into concurrency on the iPhone to help speed up your interfaces and make your apps snappier. If you have some performance bottlenecks in your app, this chapter will be very useful.

Chapter 2 brings Claus Höfele showing you how to use some desktop tools to streamline your game content pipelines. He shows you a specific example from his own extensive game developer experience, but the concepts he elucidates are applicable to many type of apps.

In Chapter 3, Ben Kazez recounts some of the lessons learned and design choices made in developing the very popular Flight Track app. He sheds some light on the process of finding and utilizing external data providers. With so much data available to your applications these days, the concepts shown here will be very helpful.

Saul Mora reminds that testing is important in Chapter 4. He shows how to use unit testing to greatly improve your code stability and help speed up your iPhone development processes.

If you were curious how computers can recognize human faces, then Chapter 5 where Leon Palm's takes on computer vision will quench that thirst. Leon introduces you to the exciting world of computer vision and shows you how to integrate the very powerful OpenCV libraries into your applications. If you are thinking of doing some Augmented Reality in your apps, this chapter will be invaluable.

If you have ever tried to render fonts in OpenGL then you know it is a complex beast. Scott Penberthy breaks it down in Chapter 6. Scott provides some tools and direction that make custom font rendering so easy that you won't go back to boring system fonts ever again.

In Chapter 7, Ben Britten Smith dips his toes into the Unity3D game engine and shows you how to leverage that middleware to build some very complex 3D games very quickly.

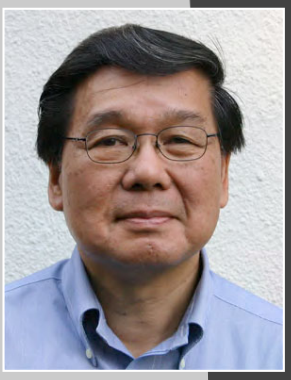
If 3D isn't your thing, head to Chapter 8 where Chuck Smith gives you a great introduction to 2D game development with the very popular framework: Cocos2d. Chuck shows you everything you need to know to get started slinging sprites around like the pros.

In Chapter 9, David Smith gives some insight into his popular Audiobooks app, and shows you how to handle lengthy audio content in his sample code. Properly dealing with large audio files is a complicated task, but David makes it so easy.

In the final chapter, Chapter 10, Arne de Vries and Joost van de Wijgerd team up to tell you about their experiences integrating push notifications into their popular eBuddy application.

iPhone and iPad development have come a long way in the short years since the SDK became available. Even working on iPhone projects every day, I still have a hard time keeping up with all of the new features and APIs available to us as iPhone developers. This book is such a great resource you will want to keep it close at hand when you embark on your next iPhone project.

Ben Britten Smith



Danton Chin

Company: ***iPhone Developer Journal (http://iphonedeveloperjournal.com/)***

Location: ***Pelham, NY***

Former Life as a Developer: ***I have programmed with both procedural and object-oriented languages on hardware ranging from mini-computers, workstations, personal computer systems, and mobile devices. I started to program in BASIC and C. In 1993, I was looking for a better way of designing and developing systems and came across NeXTStep. The night I was ready to place my order for my own NeXT workstation, NeXT announced that they were out of the hardware business. Four NeXTStep conferences and two years later it was over. Later that year, the alpha version of Java was released and over the following years I was able to watch and use a new computer language as it was born, evolved, and grew. Along the way, I got my first experience with mobile application development using J2ME (now Java ME) for Palm PDA devices. I have also developed with Actionscript and MXML, and worked with relational databases and application servers. I have worked in the financial services sector for banks and brokerage firms as well as energy, radio station, and newspaper companies.***

Life as an iPhone Developer: ***Doing iPhone development has led me to speak at the 360iDev iPhone Developer Conferences (http://www.360iDev.com/) in San Jose (http://www.360idev.com/about/san-jose-2009) and Denver (http://www.360idev.com/about/denver-2009), and at meetings of the NY iPhone Software Developers Meetup Group (http://www.meetup.com/newyork-iphone-developers/calendar/11630710/). I also started the iPhone Developer Journal blog and continue to do freelance iPhone development. I am currently working on an application for a newspaper company that should be in the App Store by the time this book is in print.***

App on the App Store:

- ***PBN (Providence Business News)***

What's in This Chapter: ***This chapter looks at concurrency solutions that are available on iPhone and iPad devices. A real-world poorly performing application is developed. Then possible approaches to a concurrent solution are discussed. A working solution using operation queues and operation objects is developed. Finally, a solution is developed using operation queues, operation objects, and blocks. The main thesis is that using a concurrent solution that makes use of operation queues, operation objects, and blocks is an optimal way of writing your application today to reduce the complexity of developing a solution with concurrency and to take advantage of changes in the iPhone OS and underlying hardware tomorrow.***

Key Technologies:

- ***Concurrency***
- ***NSOperationQueue***
- ***NSOperation***
- ***NSInvocationOperation***
- ***Blocks***

Chapter 1

Using Concurrency to Improve the Responsiveness of iPhone and iPad Applications

You do not have to have a lot of experience developing iPhone applications before you begin to realize that you may need to fetch data from a server on the Internet or that you have a CPU intensive calculation that freezes your application and prevents your user from interacting with the user interface of your application. On any platform with any computer language, the standard way of dealing with such issues is to perform these tasks in the background allowing your application's user interface to remain responsive to a user's interaction with your application. Fortunately, iPhone OS, like its much bigger sibling Mac OS X, provides a rich array of concurrency solutions for developers needing to use them. However, as you will see the concurrency solutions vary quite a bit in terms of their degree of complexity, level of abstraction, and scalability. This chapter is a brief survey of the concurrency solutions available and you will develop solutions with some of them. There is a definite point of view that I've developed by working with the iPhone SDK and trying to divine the path that Apple might take in the future that hopefully will come across. After all, whether the application is for an iPod Touch, iPhone, or an iPad it isn't cool if the application is sluggish!

Prepare for Concurrency

It had been quite a while since I had attended a conference where Steve Jobs would normally be expected to appear and attendees would go home with their cube-shaped box of books and software for the latest version of the NeXTStep operating system. When NeXT faded away many hopes were dashed, but what is happening with Mac

desktop, laptop, iPhone, iPod Touch, and iPad devices is far, far sweeter! Therefore, it was almost but not quite déjà vu as I sat in the audience at Apple's World Wide Developers Conference in 2009. As I listened, a point of view started to develop. Bertran Serlet and Bud Tribble were starting off the conference after the keynote with an overview of all the sessions (Session 700 WWDC Sessions Kickoff). Two new technologies being introduced in Snow Leopard—Grand Central Dispatch (GCD) and OpenCL (Open Computing Language)—stuck out. Grand Central Dispatch is a technology that has several facets: changes to the Mac OS X kernel; a language extension to Objective-C, C, and C++ called Blocks; and new APIs to take advantage of GCD using blocks, queues, and operation objects. OpenCL specifies OpenCL C, which is used to rewrite the calculation intensive portions of an application into C-like functions called OpenCL kernels. The OpenCL kernels are compiled at run-time for the target GPUs for execution. It was hard not to think that this was pretty amazing.

It was the "Seeker" demo (13:46 minutes into the presentation) that drove it home. Seeker is an interactive, 3D solar system simulator developed by Software Bisque (`http://www.bisque.com/`) for exploring our solar system. In the demo given by Richard S. Wright (co-author of the OpenGL SuperBible) the Seeker program calculates the position of satellites in orbit around the Earth using hundreds of calculations per frame per satellite and is able to perform the display at about 23 fps. Adding the display of space junk objects to the display of satellites brought the total number of objects in orbit around the Earth to over 12,000 maxing out the CPU and bringing the display rate down to 5 fps. Turning GCD on distributed the computations over all the cores and brought the framerate up to 30 fps! Then, GCD and OpenCL were used to display the position of over 400,000 asteroids in addition to the satellites and junk objects achieving a framerate of 30 fps. It was some demo!

At WWDC and in the months afterwards, I speculated and talked about whether GCD or some of its components, such as blocks and OpenCL, would someday become an integral part of iPhone OS in my presentations at conferences. As a follower of Apple and Apple-related news (see Table 1–1), I was aware of Apple's acquisition of P.A. Semi, a semiconductor design firm, in April 2008 and Imagination Technologies' desire to hire OpenCL engineers in December 2008 as reported by the media.

Table 1–1. *Apple and Apple-related News Items that May Impact Concurrency*

Date	News Item
2008	
April	Acquisition of P. A. Semi by Apple.
December	Imagination Technologies job openings for OpenCL engineers.
2009	
June	■ WWDC Apple announces GCD and OpenCL for Snow Leopard. ■ iPhone 3GS released.
July	Plausible Labs releases PLBlocks for Mac OS X 10.5+ and iPhone OS 2.2+.
September	■ Snow Leopard released. ■ ARM announces availability of dual core Cortex-9 ARM reference implementation.
2010	
January	■ Imagination Technologies announces the availability of the PowerVR SGX545 mobile GPU which provides full support for OpenGL 3.2 and OpenCL 1.0. ■ Apple announces the iPad powered by a 1 GHz Apple A4 SOC.
February	Plausible Labs PLBlocks 1.1 Beta with support for Mac OS X 10.5+ and iPhone OS 2.2+ including 3.2.
April	iPhone OS 3.2 and iPad released.

The availability of a third-party implementation of blocks for iPhone OS led me to wonder not *whether,* but *when* an official implementation from Apple would be available. In addition, the availability of reference implementations of dual core ARM chips, OpenCL implementations in the latest version of Imagination Technologies' PowerVR mobile GPU chips, the availability of the iPad and iPhone SDK 3.2 by April 3 all continue to point the way to the possibility that a multicore iPhone could be available around the WWDC 2010 timeframe. If there is a multicore iPhone device then we'll need enhancements to the operating system and the Cocoa Touch classes to harness the power of those cores using GCD and blocks. With rumors of multitasking coming to the next version of the iPhone operating system there may be sweeping changes ahead. We'll all know for sure once the successor to iPhone OS 3.2 is released under an NDA.

All this was speculation then and it still is. What then is an appropriate strategy for concurrency that will take advantage of multiple cores if and when they arrive? What I realized at WWDC was that using operation objects would be that strategy as long as it met the needs of my application. And, if multicore iPhones never arrive have we lost anything by using operation objects? No, especially if it helps to reduce complexity in

your application. I think it would be really sweet and exciting to have Apple's implementation of GCD, blocks, and OpenCL on the iPhone at some point in the future—possibly in the next version of the iPhone and iPhone OS at WWDC 2010! But concurrent solutions exist already so let's look at a possible use case without using concurrency and see how various concurrent solutions can increase user satisfaction and perhaps ease development efforts.

Non-Responsive User Interfaces

Almost any long-running operation can make the user interface sluggish and unresponsive. Some examples of operations that can impact the responsiveness of your user interface are loading images or data over the Internet to be displayed in your application; manipulating images or data; parsing XML or RSS; performing a complex mathematical calculation such as finding ***n*** prime numbers, calculating pi to ***m*** decimal places, or calculating Euler's constant ***e*** to ***p*** decimal places. In addition, there are situations where a concurrent solution is typically used such as sorting a socket server as well as countless other situations. As a point of discussion for looking at concurrent solutions, you will develop a simplistic application to view the images that have made it into Flickr's top 500 *interestingness* list for the most recent day. Schematically, your Interestingness app will make RESTful HTTP requests to Flickr servers and receive a response in JavaScript Object Notation (JSON) a lightweight data format for transferring data in name—value pairs (see Figure 1–1). The images will be displayed using a UITableView managed by a UITableViewController. Interestingness is a ranking assigned to each photo using a secret algorithm patented by Flickr based on a number of factors including the number of users who added the photo to a list of favorites, origin of the clickthroughs, user assigned tags, comments, relationship between the uploader of the image and those who comment on the image, and other secret factors. Flickr's `flickr.interestingness.getList` API is used to retrieve this list and does not require user authentication—only an API key. The roadmap that you will follow to build the first version of the Interestingness app will be:

- Build the user interface.
- Add a JSON parsing framework to the application.
- Compose a RESTful request to fetch a list of interestingness images.
- Use the RESTful request and the JSON framework to parse the response.
- Implement the UITableViewDataSource Protocol Methods to display the results.

Let's get started!

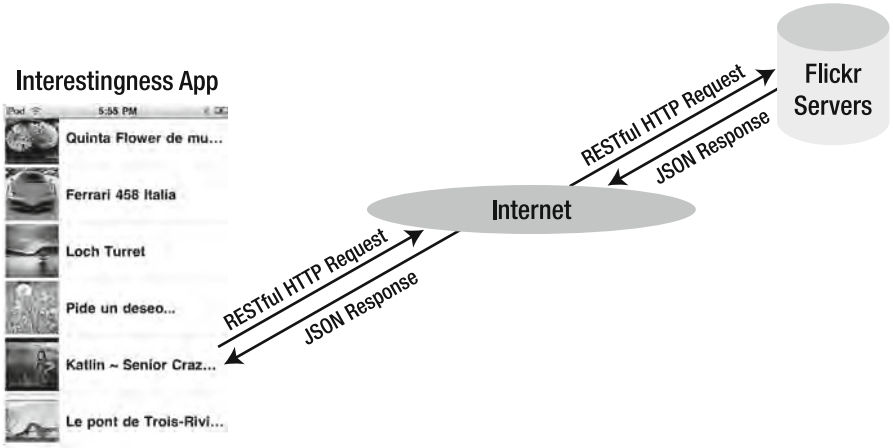


Figure 1–1. *Request/Response data flow between the Interestingness app and Flicker servers*

> **NOTE:** The projects were all built using both iPhone OS 3.1.3 and Xcode 3.2.1 and iPhone OS 3.2 and Xcode 3.2.2 on Snow Leopard.

Building the Interestingness User Interface

The application as you'll build it does not check if the network is available nor does it provide any user feedback on the progress of fetching data over a network connection. Of course, a shipping application should do both. The Reachability APIs that are a part of the SystemConfiguration.framework will allow you to check for network availability while progress indicators are a part of the standard user interface components. These aspects have been left out to focus on concurrency. So start up Xcode and create a new project using the Window-based Application template. Name the application `Interestingness` to create the header and implementation files for the `InterestingnessAppDelegate` and the main window nib file. You do not need the main nib file in this simple application and it could even be deleted but you will just leave it.

> **TIP:** For more details on creating iPhone applications, see the highly regarded book *Beginning iPhone 3 Development: Exploring the iPhone SDK*, by Jeff Lamarche and David Mark (Apress, 2009).

Next `CTRL-Click` on the Classes folder to add a new file, click on Cocoa Touch class templates, and choose the `UIViewController` subclass template making sure that the Options checkbox for `UITableViewController` is checked. Click the Next button and name the subclass `InterestingnessTableViewController`. Make sure that the checkbox to create the header file is checked then click the Finish button (see Figure 1–2).

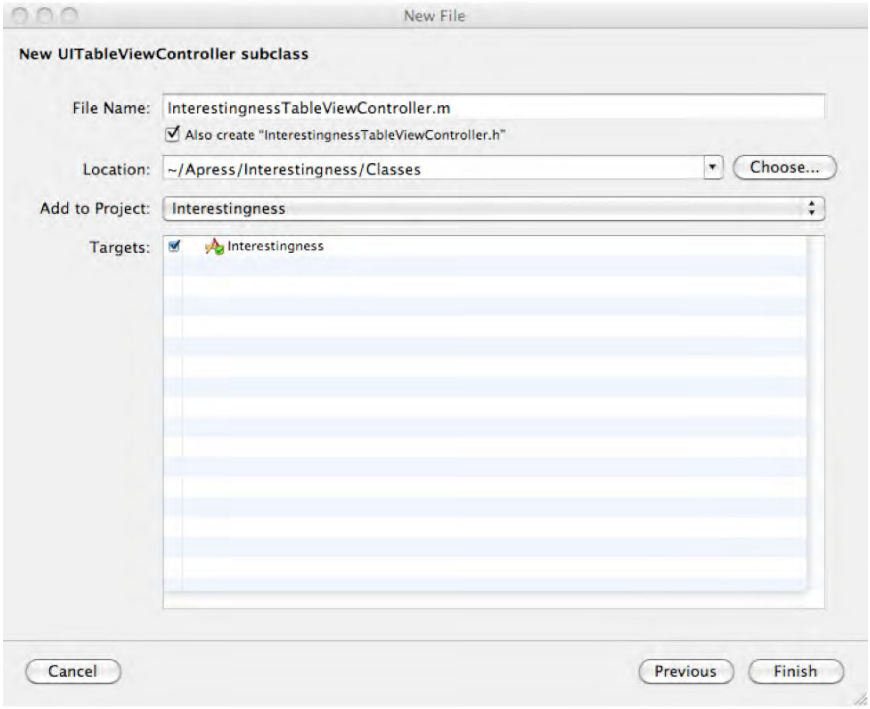


Figure 1–2. *Subclassing UITableViewController*

In the application delegate header file, you forward declare `InterestingnessTableViewController` and declare the variable for an instance of this subclass and name it `tableViewController` so that the declaration of the application delegate class appears as in Listing 1–1.

Listing 1–1. *InterestingnessAppDelegate Header fFle*

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

@class InterestingnessTableViewController;

@interface InterestingnessAppDelegate : NSObject <UIApplicationDelegate> {
        UIWindow *window;
        InterestingnessTableViewController *tableViewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;

@end
```

On the implementation side of the application delegate import the header file for the `InterestingnessTableViewController` and in the `applicationDidFinishLaunching` method, allocate and initialize an instance of our table view controller, set the frame for the view controller, add the table view controller's view as a subview of the window, and for good memory management practice release the `tableViewController` in the `dealloc` method as in Listing 1–2.

Listing 1–2. *InterestingnessAppDelegate Implementation File*

```
#import "InterestingnessAppDelegate.h"
#import "InterestingnessTableViewController.h"

@implementation InterestingnessAppDelegate

@synthesize window;


- (void)applicationDidFinishLaunching:(UIApplication *)application {

        tableViewController = [[InterestingnessTableViewController alloc]
                                        initWithStyle:UITableViewStylePlain];

        [[tableViewController view] setFrame:[[UIScreen mainScreen] applicationFrame]];

        [window addSubview:[tableViewController view]];

    [window makeKeyAndVisible];
}


- (void)dealloc {
        [tableViewController release];
    [window release];
    [super dealloc];
}


@end
```

Note that you are programmatically creating your table view controller and that you are not using a nib file in which a table view is defined with a data source and a delegate. In this case, the `UITableViewController` by default sets the data source and the delegate to `self`. Out of the box, the subclass provides stubs for two of the required methods of the `UITableViewDataSource Protocol`—`tableView:cellForRowAtIndexPath:` and `tableView:numberOfRowsInSection:`—and one of the optional methods `numberOfSectionsInTableView:`. In addition, the subclass provides a stub implemention of only one of the optional methods— `tableView:didSelectRowAtIndexPath:`—of the `UITableViewDelegate Protocol` which you do not need to implement in this case since the Interestingness app will not be providing a detail view for the selected row. Since you are creating the table view controller programmatically uncomment the `initWithStyle:` method. At this point, that is the only change in the implementation of `InterestingnessTableViewController`. So go ahead and build and run this in the Simulator to make sure that everything works. You currently have a responsive but empty table view!

NOTE: The Interestingness project up to this point is in the folder Interestingness-Version1.

Adding A JSON Parsing Framework to the Interestingness App

As was mentioned earlier the Interestingness app will receive a response to an HTTP request in JSON format (see Figure 1–1). In order to use the data, the reponse will be parsed using a JSON parser. JSON has become increasingly popular as a data exchange format and parsers are available for just about every computer language. The next step then is to download and add a JSON parsing framework to your project. The framework used in this project is the json-framework which can be downloaded from Google's Code repository at `http://code.google.com/p/json-framework/` (see the "Resources" section). Download and expand the disk image. Drag the JSON folder to the `Interestingness` project and drop it on the project or into your favorite folder (such as Classes). Be sure to check the checkbox to copy the items to the destination folder (Figure 1–3). You can also CTRL-Click to add an existing folder and its contents to the project making sure to select the option to copy the items to the destination folder. Now importing `JSON.h` will provide access to the methods that make up the json-framework.

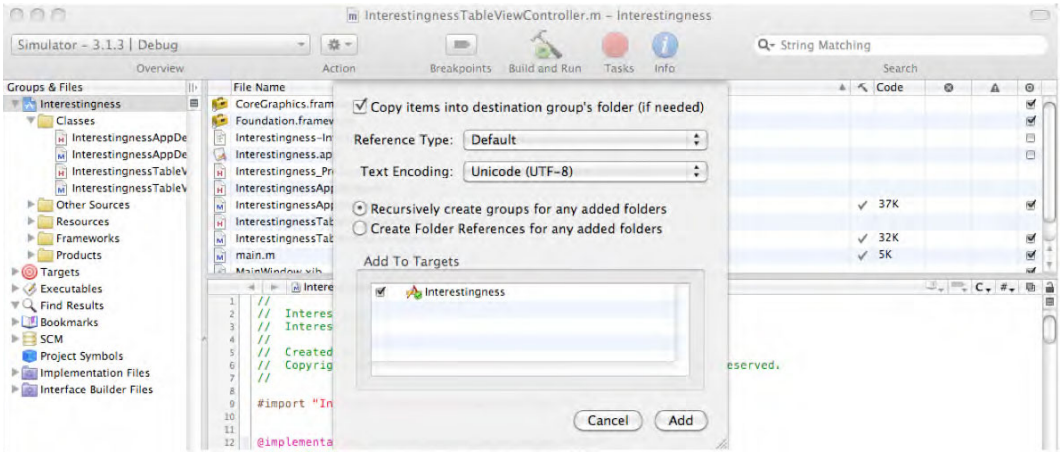


Figure 1–3. *Adding the JSON classes to the Interestingness project*

Composing a RESTful Request for a List of Interestingness Images

To fetch the images from the Flickr `interestingness` API make a RESTful request to Flickr for a list of these images. The information needed to build a URL for the individual images is extracted from the list to build a URL for an individual image. A Flickr API request consists of four components—a service endpoint, a method name, an API key, and a list of required and optional parameters for the method:

- *Service Endpoint*: Flickr accommodates requests and provides responses via three formats: REST, XML-RPC, and SOAP. Flickr also provides responses using JSON and PHP. You will be making RESTful requests using HTTP GET for which the service endpoint is `http://api.flickr.com/services/rest/`.
- *Method name*: Flickr provides authenticated and nonauthenticated access to the photos and the extended data attributes around the photos that are uploaded to their site. Most of the APIs require authentication in addition to an API key. You will use a nonauthenticated, "public" API named `flickr.interestingness.getList` to get a list of the most interesting photos.
- *Flickr API key*: If you have a Flickr API key you will need it in order to download images for your project. If you do not have a Flickr API key, head over to Flickr (`http://www.flickr.com/services/apps/create/apply/`) to create an API key for which you will need a Yahoo account.
- *Method parameters*: As parameters you will need to provide:
  - per_page (optional): The number of photos to return per page.
  - format (optional): Specify that the response be in JSON format. Additionally, you just want the raw JSON output without the enclosing function wrapper so that as part of the request one of the parameters of the request will be nojasoncallback=1.

Your Flickr request will appear as follows:

```
http://api.flickr.com/services/rest/?method=flickr.interestingness➥
.getList&api_key=%@&tags=&per_page=%d&format=json&nojasoncallback=1
```

with two values to be filled in, the API key and the number of photos per page, which will be done when you create the NSString.

Now to make the changes to the Interestingness application so it fetches and displays the images change `InterestingnessTableViewController.h` as follows:

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

@interface InterestingnessTableViewController : UITableViewController {

        NSMutableArray *imageTitles;
        NSMutableArray *imageURLs;
}
-(void)loadInterestingnessList;

@end
```

This declares two mutable arrays to store the image names and URLs and a method to load the list of interestingness images from Flickr.

Next, you will need to change the implementation. Begin by importing JSON.h, defining your Flickr API key, modifying the `initWithStyle:` method to initialize the two mutable arrays, and uncomment the `viewWillAppear:` method so that you can add the call to `self` to load the list of images:

```
#import "InterestingnessTableViewController.h"

#import "JSON.h"

#define API_KEY @"INSERT YOUR FLICKR API KEY HERE"

@implementation InterestingnessTableViewController


- (id)initWithStyle:(UITableViewStyle)style {
    if (self = [super initWithStyle:style]) {
                imageTitles = [[NSMutableArray alloc] init];
                imageURLs  = [[NSMutableArray alloc] init];
}
    return self;
}

- (void)viewWillAppear:(BOOL)animated {
        [super viewWillAppear:animated];
        [self loadInterestingnessList];
}
```

Now is as good a time as any to remember to release the mutable arrays in the `dealloc` method so that you don't leak memory.

```
- (void)dealloc {
        [imageTitles release];
        [imageURLs release];
    [super dealloc];
}
```

Using the RESTful Request and the JSON Parser to Parse the Response

Now implement the `loadInterestingnessList` method:

```
-(void)loadInterestingnessList
{

    NSString *urlString = [NSString stringWithFormat:@"http://api.flickr.com/services↩
/rest/?method=flickr.interestingness.getList&api_key=%@&extras=description&tags=↩
&per_page=%d&format=json&nojsoncallback=1", API_KEY, 500];

    NSURL *url = [NSURL URLWithString:urlString];

    NSError *error = nil;

    NSString *jsonResultString    = [NSString stringWithContentsOfURL:url
                                                encoding:NSUTF8StringEncoding
                                                                    error:&error];
```

```
    NSDictionary *results = [jsonResultString JSONValue];

    NSArray *imagesArray = [[results objectForKey:@"photos"] objectForKey:@"photo"];

    for (NSDictionary *image in imagesArray) {

        // build the url to the image

        if ([image objectForKey:@"id"] != [NSNull null]) {
            NSString *imageURLString = [NSString
                stringWithFormat:@"http://farm%@.static.flickr.com/%@/%@_%@_s.jpg",
                                                    [image objectForKey:@"farm"],
                                                    [image objectForKey:@"server"],
                                                    [image objectForKey:@"id"],
                                                    [image objectForKey:@"secret"]];

            [imageURLs addObject:[NSURL URLWithString:imageURLString]];

            // get the image title

            NSString *imageTitle = [image objectForKey:@"title"];

            [imageTitles addObject:([imageTitle length] >  0 ? imageTitle ➥
: @"Untitled")];

        }

    }
}
```

Here you build the URL string using NSString's class method `stringWithFormat:` to provide the API key and the number of images you want to retrieve per page. Then you retrieve the URL with NSString's convenience method `stringWithContentsOfURL:encoding:error` and store the result in `jsonResultString`. You then use the json-framework to return the NSDictionary represented by the string and retrieve the array of dictionary objects representing the images using the key "photo". Next, iterate through the array of dictionary objects (the info for each image) using the keys farm, server, id, and secret to retrieve the corresponding value in order to build a URL for the image and add the URL to your mutable array of URLs in the instance variable `imageURLs`. Your final steps in this method are to retrieve the title for the image and store either the title if the length is greater than zero or "Untitled" in your mutable array of titles `imageTitles`.

> **NOTE:** More details on how to build a URL for an individual photo can be found at `http://www.flickr.com/services/api/misc.urls.html`.

Implementing the UITableViewDataSource Protocol Methods to Display the Results

Next, you need to modify the UITableViewDataSource methods `numberOfSectionsInTableView:`, `tableView:numberOfRowsInSection:`, and `tableView:cellForRowAtIndexPath:` to fetch and display the images in our table. Since your user interface is very basic there will only be one section in the table view and the `numberOfSectionsInTableView:` method returns the following:

```
-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
        return 1;
}
```

Next change the `tableView:numberOfRowsInSection:` method to return the number of elements in the imageURLs array:

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection↩
:(NSInteger)section {
    return [imageURLs count];
}
```

Finally, change the `tableView:cellForRowAtIndexPath:` method to set the text of the label to the title of the image, fetch the image using NSData's class method dataWithContentsOfURL, and finally set the cell's imageView to the image that was just downloaded:

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath↩
:(NSIndexPath *)indexPath {

    static NSString *CellIdentifier = @"Cell";

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

    // set the title

        [[cell textLabel] setText:[imageTitles objectAtIndex:[indexPath row]]];

        // fetch the image

        NSData *data = [NSData dataWithContentsOfURL:[imageURLs objectAtIndex↩
:[indexPath row]]];

        // set the cell's image

        [[cell imageView] setImage:[UIImage imageWithData:data]];

    return cell;
}
```

NOTE: The project up to this point is in the folder Interestingness-Version2.

Now it is time to test the Interestingness application on a device. Select the Interestingness target, CTRL-Click, select Info, and then select the Properties tab and set the Identifier to one that matches your provisioning profile that has been set up with your iPhone Developer Certificate. Then set the Active SDK to Device – 3.1.3 | Debug and build and run. The application can be installed and tested in the Simulator but performance of the application in the Simulator does not resemble the performance of the application on a real device. The Interestingness application should look like Figure 1–4 although the images and text will not be the same.

CAUTION: A splash screen will load immediately but it may take the application several seconds before the interestingness images appear.

TIP: If the application compiles but no images appear you may have forgotten to enter a valid Flickr API key so be sure to check that. Also, you must be a paid member of the iPhone Developer Program in order to install and test on an iPhone, iPod, or iPad device. See `http://developer.apple.com/programs/iphone/develop.html`.

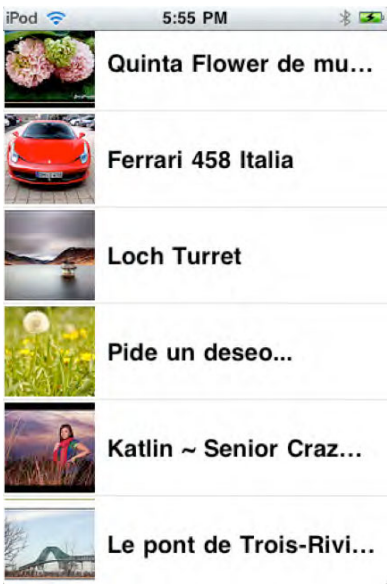


Figure 1–4. *A screen shot of the Interestingness application*

Concurrency Landscape

Why is the application slow? Both Cocoa and iPhone Applications start life with one thread and a run loop. Basically, a run loop listens for events and passes them on to an appropriate handler. Once the handler finishes processing, the event control returns to the run loop which either processes the next event or puts the main thread to sleep until there is something to do. Your application is doing all of its work on the main thread preventing it from responding to any user interface events or any other event until your task has finished.

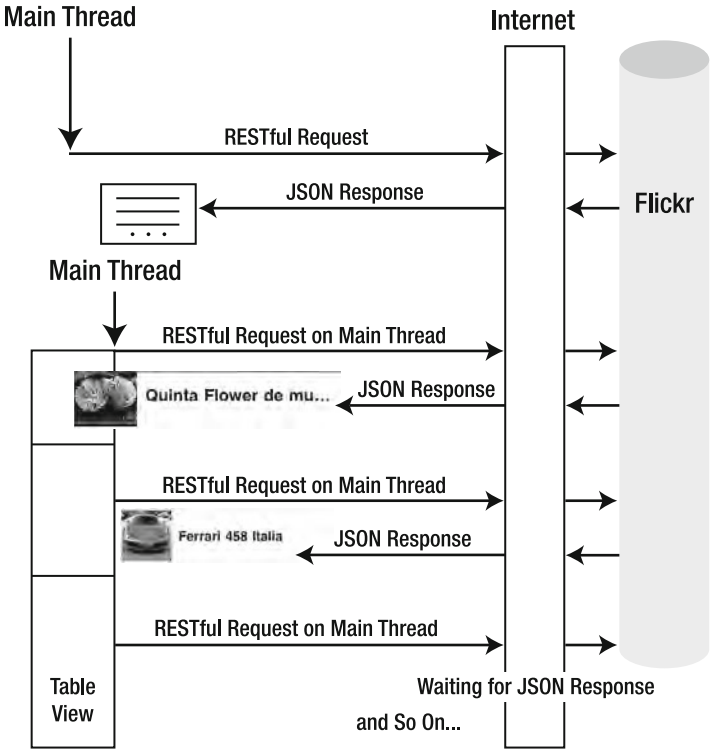


Figure 1–5. *The Interestingness app makes a RESTful request for the master list of images and then one RESTful request for each image for each exposed table view cell , all on the main thread in Version 2 of the app.*

Interestingness fetches data over the Internet at two points—once to fetch the list of Interestingness images and then one fetch for each image in each cell of the table view (see Figure 1–5). It does so synchronously, and until the data has been returned, no updates of the user interface can occur. Our application needs to perform such time consuming tasks in a background thread and not on the main thread. There are a lot of concurrent options available as you can see from Table 1–2. The concurrency solutions range from a high level of abstraction and a lower level of complexity to a low level of abstraction and a higher level of complexity. Depending upon the needs of your application one solution may be better than another. It may be almost obvious but it does not hurt to state that the best concurrency solution is the concurrency solution that

has the highest level of abstraction that is consistent with your application needs. There are multiple benefits from doing so and in this case it applies particularly to the use of operation objects—it reduces complexity in your application (KISS), it may insulate your application from lower level changes, it may lay the groundwork for future changes in the operating system, and it may allow your application to take advantage of underlying hardware changes. Using the highest level of abstraction is consistent with the history and development of computers, programming languages, and programming methodologies.

Table 1–2. *Available Concurrency Solutions As of April 2010*

Level of Abstraction	Technology	Description	iPhone OS	Mac OS X	Complexity
High	Operation objects	NSOperation NSOperationQueue	Yes	10.5+	Low
	Grand Central Dispatch	Dispatch queues and blocks, etc.	No	10.6+	
	Cocoa Threads	NSThread NSObject	Yes	Yes	
	Asynchronous methods	Some classes have both synchronous and asynchronous methods, NSURLConnection, for example	Yes	Yes	
Low	POSIX Threads (pthreads)	C-based APIs and libraries for creating and managing threads	Yes	Yes	High

Considerations When Using Concurrent Solutions

There are some general considerations to be aware of when implementing any concurrent solution. A primary consideration is that generally UIKit classes are not thread safe unless the Apple documentation specifically states that it is. All updates to the user interface should occur on the main thread and not from a background thread. Another major concern that must be taken into account once an application has two or more threads is synchronizing access to shared data, especially mutable data. Altering shared data structures from more than one thread can lead to race conditions and deadlocks.