

Extracted from:

# iOS SDK Development

This PDF file contains pages extracted from *iOS SDK Development*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2012 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The  
Pragmatic  
Programmers

# iOS SDK Development



Chris Adamson  
and Bill Dudney  
*Edited by Brian P. Hogan*

# iOS SDK Development

Chris Adamson  
Bill Dudney

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

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

The team that produced this book includes:

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

Copyright © 2012 The Pragmatic Programmers, LLC.  
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.  
ISBN-13: 978-1-934356-94-4  
Encoded using the finest acid-free high-entropy binary digits.  
Book version: P1.0—November 2012

### 4.3 Building a Detailed Recipe View

Now that we know how our view gets onscreen, let's flesh out our recipe model a bit so that we have something nicer to look at. Along the way, we'll learn a bit about some of the other subclasses of `UIView` in `UIKit`.

Open the `PRPRecipe.h` header and add a property for the directions and a property for an image of the completed recipe. The code should look like this:


```
ViewsAndControllers/Recipes_03/Recipes/PRPRecipe.h
@interface PRPRecipe : NSObject

@property(nonatomic, copy) NSString *title;
@property(nonatomic, copy) NSString *directions;
@property(nonatomic, strong) UIImage *image;


@end
```



Be sure to use `strong` rather than `copy` on the `UIImage` property: it would waste memory to copy a lot of image data around.

With these two properties in place, we can step up our game for the look of our view. Open the `PRPViewController.xib` file by selecting it in the project navigator. We want to do three things here: first, make the title use a bold font; second, add a text view so we can display the directions; and third, put an image view on the screen so we can show what the completed recipe looks like.

To change the title to use a bold font, we just need to change the font that the `UILabel` uses. Select the title label, open the Attributes inspector ()<sup>4</sup>, and click the Change Font button, as seen in [Figure 36, Changing font of a UILabel, on page 6](#).

When the pop-up window is visible, select the Font pop-up menu and then select the System Bold item.

Next, let's add a text view to display the directions. To do that, we need to go to the Object library ()<sup>3</sup>. In the filter field at the bottom of this pane, type "text" to narrow the displayed objects to just the ones with *text* in their name or description.

Click the Text View object and drag it onto the view. Using the Size inspector ()<sup>5</sup>, place it at {20, 50, 280, 100}. In the Attributes inspector ()<sup>4</sup>, turn off editing, turn off horizontal scrollers, and change the background color to clear.

Next we want to place an image view. In the Search field, type "image," select the Image View, and drag it into the view. Size the view with the Size inspector

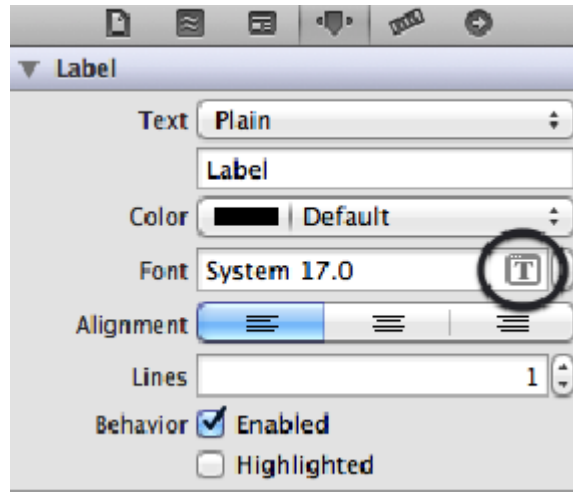


Figure 36—Changing font of a UILabel

at {20, 160, 280, 280}. The interface should look like [Figure 37, Updated interface, on page 7](#).

Now that we have some new views on the screen to show additional data, we need to wire up the controller to get the data from the model into these views. That means it's time to head back over to PRPViewController.h. Select the file in Xcode and add two new IBOutlet properties, like this:

**ViewsAndControllers/Recipes\_03/Recipes/PRPViewController.h**

```
@property(nonatomic, strong) IBOutlet UITextView *directionsView;
@property(nonatomic, strong) IBOutlet UIImageView *imageView;
```

Now that the view controller has outlets for these two elements of the view, we need to head back to the interface file to make the connections to the text and image views that we added just a couple of paragraphs back. Open PRPViewController.xib again, select File's Owner, and control-click and drag to the text view. Let up on the mouse and select directionsView from the pop-up. Do the same for the image view, but connect it to the imageView outlet. The Connections inspector should look like [Figure 38, Updated connections, on page 7](#).

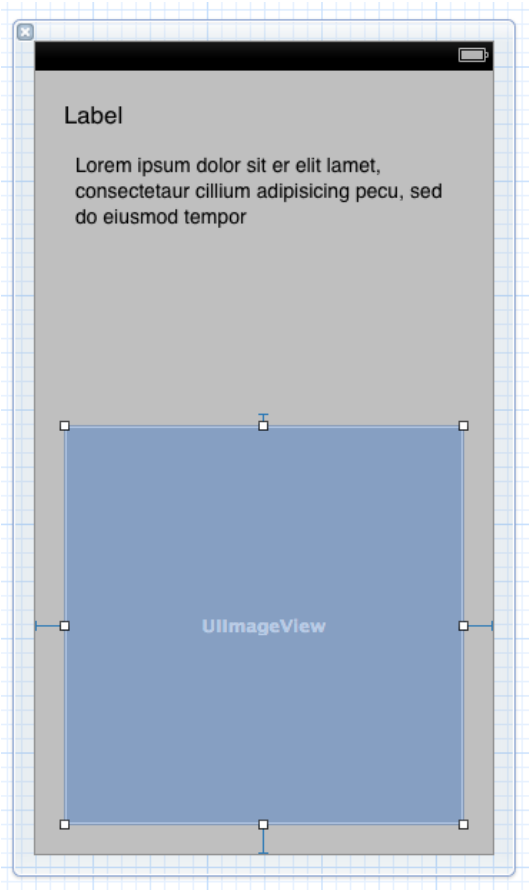


Figure 37—Updated interface

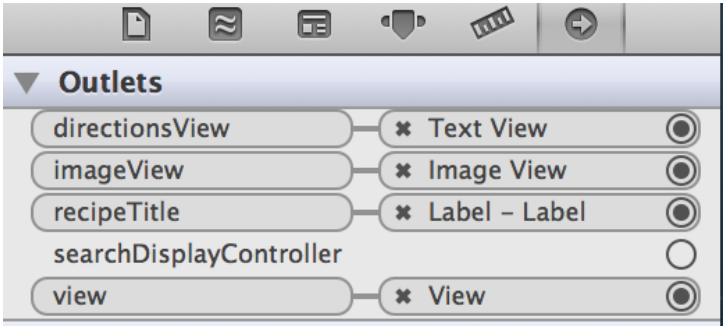


Figure 38—Updated connections

Now we move on to updating the code in our controller to put the data we get from the model into the view. Open the `PRPViewController.m` and go back to the `viewWillAppear:` method. We need to grab the directions text from the recipe and place that into the `directionsView`, and we need to grab the image of the completed recipe and place that into the `imageView`. The code is very simple; just rewrite `viewWillAppear:` like this:

```
ViewsAndControllers/Recipes_03/Recipes/PRPViewController.m
Line 1 - (void)viewWillAppear:(BOOL)animated {
2     [super viewWillAppear:animated];
3     self.recipeTitle.text = self.recipe.title;
4     self.directionsView.text = self.recipe.directions;
5     if(self.recipe.image) {
6         self.imageView.image = self.recipe.image;
7     }
8 }
```

This code is not much different from our previous version of `viewWillAppear:`. On line 4 we put the directions string from the recipe into the text property of the `directionsView`, which simply places that string into the text view.

Next, on lines 5–7 we put the image from the recipe into the `imageView`'s image property. We replace the `imageView`'s image only if the recipe's image is set.

### Sizing Images

Images are a big part of making iOS applications stand out, but they can be a huge performance drain. An often-overlooked performance issue is the expense of compressing an image. In the sample code the image of the chocolate chip cookies is 280 x 280. The image was purposefully sized that way to match the size of the `UIImageView` instance in our view. If we were to try to display a 5-megapixel image in a 280 x 280 image view, it would display just the same but the image view would have to do a bunch of work to make the image fit into that 280 x 280 size. In essence it would have to read five million points from the image to fill seventy-eight thousand points on the screen. That is very wasteful, both in terms of performance (how long it takes) as well as power consumption (how much power is drained from the battery). On the iPhone 4 you might not notice the performance hit because it's so fast and we are only displaying one image, but our users would notice the battery drain.

We are done with the heavy lifting, but we still need to put an image and directions into the recipe model object. Remember that we created an instance of `PRPRecipe` in the `PRPAppDelegate`'s `application:didFinishLaunchingWithOptions:` method. So let's go back there and update that code. Open the file `PRPAppDelegate.m` and select the `application:didFinishLaunchingWithOptions:` method.



The downloadable sample code has an image and some directions for chocolate chip cookies. Feel free to do something similar or just copy what is in the sample code. To add an image file to Xcode, you can drag it from the Finder to the project navigator. When prompted, make sure to check the “Copy items into destination group’s folder (if needed)” check box; this keeps the image file with your source files rather than just managing a reference to the original image on your filesystem. Don’t forget to resize and crop the image to be 280 x 280.

The updated code should look like this:

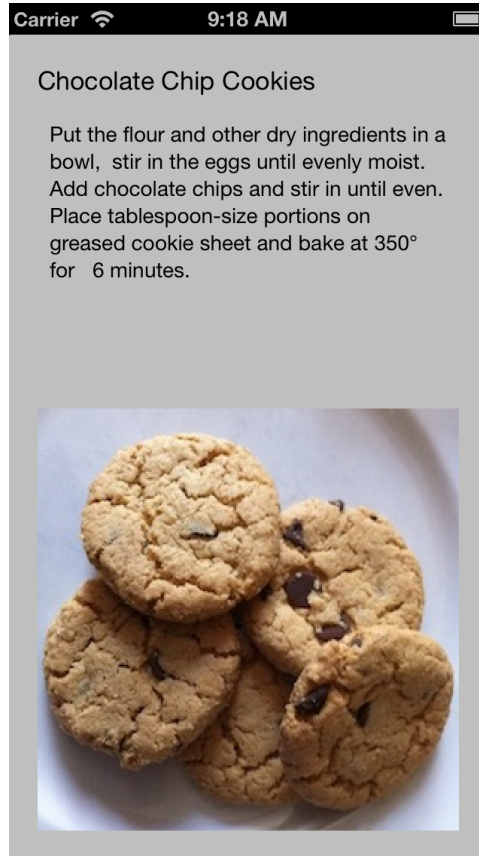
```
ViewsAndControllers/Recipes_03/Recipes/PRPAppDelegate.m
Line 1 - (BOOL)application:(UIApplication *)application
- didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
-   NSString *directions = @"Put the flour and other dry ingredients in a bowl,\
-   stir in the eggs until evenly moist. Add chocolate chips and stir in until even. \
5   Place tablespoon-size portions on greased cookie sheet and bake at 350° for \
-   6 minutes.";
-   PRPRecipe *recipe = [[PRPRecipe alloc] init];
-   recipe.title = @"Chocolate Chip Cookies";
-   recipe.image = [UIImage imageNamed:@"cookies.jpg"];
10  recipe.directions = directions;
-   self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen] bounds];
-   self.viewController = [[PRPViewController alloc]
-                           initWithNibName:@"PRPViewController"
-                           bundle:nil];
15  self.viewController.recipe = recipe;
-   self.window.rootViewController = self.viewController;
-   [self.window makeKeyAndVisible];
-   return YES;
- }
```

The directions string is set up on line 6. The backslash character (“\”) tells the compiler that the string continues on the next line.<sup>1</sup> On line 9, we create a UIImage from the IMG\_1948.jpg image. Feel free to use this image, or take a break from coding and go bake your own cookies and take a picture of them.

Now that our app is complete, click the Run button in Xcode. The running app should look like [Figure 39, Completed recipes app, on page 10](#).

Congratulations again! You have a view controller that is capable of showing a recipe, including a photo of what the completed product will look like. Before we go on to the next topic, we need to take a short detour into understanding how view controllers help us to manage the memory used by its view.

1. We use this character to suit the formatting of the book; you can keep the string all on one line in your own code.



---

**Figure 39—Completed recipes app**

---