VISUAL QUICKSTART GUIDE

Get up and running in no time!



With downloadable code samples!

iPhone Application Development for iOS 4

DUNCAN CAMPBELL

LEARN THE QUICK AND EASY WAY!

iPhone Application Development

FOR IOS 4

DUNCAN CAMPBELL



Peachpit Press

Visual QuickStart Guide iPhone Application Development for iOS 4

Duncan Campbell

Peachpit Press 1249 Eighth Street Berkeley, CA 94710 510/524-2178 510/524-2221 (fax)

Find us on the Web at www.peachpit.com. To report errors, please send a note to errata@peachpit.com. Peachpit Press is a division of Pearson Education.

Copyright © 2011 by Duncan Campbell

Editor: Whitney Walker and Cliff Colby Production Coordinator: Danielle Foster Copyeditor/proofreader: Kim Wimpsett Technical Editor: James Sugrue Compositor: Danielle Foster Indexer: Valerie Perry Cover Design: RHDG/Riezebos Holzbaur. Peachpit Press Logo Design: MINE[™] www.minesf.com Interior Design: Peachpit Press

Notice of Rights

All rights reserved. No part of this book may be reproduced or transmitted in any form by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. For information on getting permission for reprints and excerpts, contact permissions@peachpit.com.

Notice of Liability

The information in this book is distributed on an "As Is" basis, without warranty. While every precaution has been taken in the preparation of the book, neither the author nor Peachpit shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer software and hardware products described in it.

Trademarks

Visual QuickStart Guide is a registered of Peachpit Press, a division of Pearson Education. Any other product names used in this book may be trademarks of their respective owners.

Apple, Cocoa, Cocoa Touch, Dashcode, iPhone, iPod touch, Safari, and Xcode are trademarks of Apple Inc. registered in the U.S. and other countries.

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 Peachpit was aware of a trademark claim, the designations appear as requested by the owner of the trademark. All other product names and services identified throughout this book are used in editorial fashion only and for the benefit of such companies with no intention of infringement of the trademark. No such use, or the use of any trade name, is intended to convey endorsement or other affiliation with this book.

ISBN-13: 978-0-321-71968-3 ISBN-10: 0-321-71968-9

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

Dedication

For my son, Hamish.

Acknowledgments

Thanks to *Whitney Walker, Clifford Colby, Kim Wimpsett, Danielle Foster, Valerie Perry*, and everyone else at Peachpit Press who worked so hard to make this book happen.

Thanks to James Sugrue for his technical-editing expertise.

A big thank-you to my good friend *Kane Nickolichuck* who all those years ago pestered me relentlessly into buying my first Macintosh computer.

Cuddles and pets to my dog, *Kip*, for again keeping me company during the cold (yes, even in Australia!) winter evenings I spent working on this book.

Finally, the biggest thanks go to my wife, *Sarah*, for single-handedly looking after our newborn son while I spent the evenings locked away in my office each night.

Contents at a Glance

	Introduction
Chapter 1	Objective-C and Cocoa
Chapter 2	The iPhone Developer's Toolbox
Chapter 3	Common Tasks
Chapter 4	iPhone User Interface Elements
Chapter 5	Tabs and Tables
Chapter 6	Files and Networking
Chapter 7	Touches, Shakes, and Orientation
Chapter 8	Location and Mapping
Chapter 9	Multimedia
Chapter 10	Contacts, Calendars, E-mail, and SMS 405
Chapter 11	Multitasking
	Index

This page intentionally left blank

Table of Contents

	Introduction
Chapter 1	Objective-C and Cocoa
	Frameworks
	Classes
	Methods
	Creating objects
	Properties
	Memory Management
	Autorelease pools
	Commonly Used Classes
	Strings
	Dates and times
	Arrays
	Dictionaries
	Notifications
	Timers
	Design Patterns
	Model View Controller
	Delegate
	Target-Action
	Categories
	Singletons

Chapter 2	The iPhone Developer's Toolbox
	About the Xcode IDE
	About the Groups & Files pane
	Targets
	About the toolbar
	About the details pane
	About the editor pane
	Gutter and focus ribbon
	Find-and-replace operations
	Bookmarks
	Jump-to-definition and help 53
	Code completion
	About the navigation bar
	Creating new files
	Building and running your application
	Cleaning
	About the iPhone Simulator 61
	About Interface Builder
	About the document window 65
	About the Library window 67
	About the inspector window 67
	About the Documentation
	The Xcode Organizer
	Projects & Sources
	Devices
	iPhone Development

Chapter 3	Common Tasks	83
	Application Startup and Configuration	84
	Using the application delegate	84
	Understanding application settings	87
	Working with user preferences	87
	Application preferences	90
	Adding controls	92
	Localization	94
	Accessibility	98
	Making your applications accessible	99
	Accessibility attributes	101
	Interapp Communication	03
	Sharing information between applications	05
	Using the pasteboard	09
Chapter 4	iPhone User Interface Elements 1	11
	Views	112
	Frames	112
	Bounds	113
	Animation	115
	Autosizing	117
	Custom drawing	118
	Transforms	23
	Image Views	26
	Animating images	127
	Scrolling	29

Zoom
Paging
Labels
Progress and Activity Indicators
Indicating progress
Showing activity
Alerts and Actions
Alerting users
Confirming an action
Picker Views
Toolbars
Toolbar items
Text
To use keyboards:
Restricting content
Text views
Data detectors
Hiding the keyboard
Scrolling the interface
Web Views
Running JavaScript
Loading local content and handling hyperlinks168
Controls
Buttons
Switches
Sliders
Segmented controls

Chapter 5	Tabs and Tables
	View Controllers
	Presenting views
	Responding to changes in orientation
	Displaying modal views
	Handling low-memory conditions
	Tab Views
	Adding graphics and titles to tabs
	Table Views
	Grouping rows into sections and styles 204
	Editing and searching table views
	Drilling down in table views
	Creating custom cells
Chapter 6	Files and Networking
Chapter 6	Files and Networking .233 Files .234 The file system .236 Common directories .237 Working with files .239 Previewing documents .244 Networking .248 Retrieving content from web pages .248
Chapter 6	Files and Networking.233Files.234The file system.236Common directories.237Working with files.239Previewing documents.244Networking.248Retrieving content from web pages.248Parsing XML.254
Chapter 6	Files and Networking
Chapter 6	Files and Networking

Chapter 7	Touches, Shakes, and Orientation
	Touch
	Adding tapping support
	Adding long-touch support
	Multi-Touch Gestures
	The iPhone Accelerometer
	Detecting shakes
	Determining orientation
	Redrawing the interface when the orientation changes
	Responding to the accelerometer
Chapter 8	Location and Mapping
	About Core Location
	Handling location updates
	Testing outside the simulator
	Increasing the accuracy
	Adding a timeout
	Accessing the compass
	About Map Kit
	Map Overlays
	Adding annotations
	Adding reverse geocoding
	Putting It All Together
Chapter 9	Multimedia
	Playing Audio
	Providing more control

	Responding to audio events
	Playing audio in the background
	Controlling audio from the background
	Recording Audio
	Using the iPhone's Camera
	Taking photos and video
	Playing Video
	To gain more control over movie playback 386
	Using the iPod Library
	Accessing media items
	Accessing media collections
	Using the media picker
	Playing media
Chapter 10	Contacts, Calendars, E-mail, and SMS 405
Chapter 10	Contacts, Calendars, E-mail, and SMS
Chapter 10	Contacts, Calendars, E-mail, and SMS
Chapter 10	Contacts, Calendars, E-mail, and SMS
Chapter 10	Contacts, Calendars, E-mail, and SMS
Chapter 10	Contacts, Calendars, E-mail, and SMS
Chapter 10	Contacts, Calendars, E-mail, and SMS .405 Working with the Address Book 406 Group records .410 Person records .411 Adding a User Interface .418 Picking people .418 Editing people .421
Chapter 10	Contacts, Calendars, E-mail, and SMS
Chapter 10	Contacts, Calendars, E-mail, and SMS
Chapter 10	Contacts, Calendars, E-mail, and SMS
Chapter 10	Contacts, Calendars, E-mail, and SMS.405Working with the Address Book406Group records.410Person records.411Adding a User Interface.418Picking people.418Editing people.421The iPhone Calendar.430Viewing event details.438
Chapter 10	Contacts, Calendars, E-mail, and SMS

Chapter 11	Multitasking	. 455
	What Is Multitasking?	456
	Entering and exiting background mode	457
	Multitasking services	459
	Responding to Local Notifications	466

Index	•	•	•	•	•		•		•	•	•		•	•		•	•	•	•		•	•	•	46	9
-------	---	---	---	---	---	--	---	--	---	---	---	--	---	---	--	---	---	---	---	--	---	---	---	----	---

Introduction

Welcome to the updated version of this Visual QuickStart Guide for iPhone application development.

A lot has happened since the last version of this book was published: In only one short year, not only have we seen the introduction of the revolutionary iPad, but we've also seen the all-new iPhone 4, with its gorgeous high-resolution display and powerful new hardware capabilities.

The tools for iPhone development have also had a major upgrade. iOS 4 brings with it many new application programming interfaces (APIs) that give developers even more access to the iPhone's underlying hardware, as well as adds exciting new capabilities, such as multitasking and highdefinition (HD) video recording and editing. At the time of this writing, more than 250,000 applications are available for download from the iTunes App Store, with more being added every minute—it's an exciting time to be an iPhone developer!

This book is geared mainly toward new iPhone developers, but you should have some prior knowledge of a C-based language and be familiar with object-oriented (OO) concepts. It would take a book many times this size to cover all of the iPhone software development kit (SDK), so I focus on some of the more common and interesting subjects I think you should know about when developing your own iPhone applications.

How to Use This Book

I find that I always learn better by example, so I have created stand-alone applications when demonstrating the concepts in the book. The aim is to give you enough information to get you started coding (and building something useful) and then point you to the relevant place in the documentation for more information.

You should be able to jump straight into a chapter and start coding without reading the prior chapters, but if you are a beginner, I recommend you read the first few chapters, which discuss the tools and language used for iPhone development.

This book is a Visual QuickStart Guide, so it's filled with images to walk you through what you'll see on your computer screen as you build your iPhone applications. However, the interfaces for most of the examples are created directly in code, rather than by using Interface Builder. You might think this is unusual, since Apple has provided you with a powerful tool that makes laying out your application's user interface quick and easy, but it's important that you first learn what's happening under the hood. This will make it much easier for you to figure out where to look when things aren't working the way they should.

The source code for all the examples in this book—more than 65 projects—is available as a free download from my Web site:

http://objective-d.com/iphonebook/

I strongly encourage you to check them out.



iPhone User Interface Elements

iOS offers a rich set of buttons, sliders, switches, and other user interface elements for you to use in creating your applications. These elements can be roughly divided into two main groups, views and controls.

Views provide the primary canvas and drawing functionality of your user interface. They also give your application the ability to handle touch events.

Controls extend upon this functionality and provide a way for users to interact with your application by defining what is known as the *target-action* mechanism: the ability for a control to send an *action* (method call) to a *target* (object) when an *event* (touch) occurs.

In this chapter, you'll look at the various views and controls available in iOS and examine how to use them.

All the examples use the View-based Application template, with the code running in the view controller.

In This Chapter

Views	112
Image Views	126
Scrolling	129
Labels	136
Progress and Activity Indicators	139
Alerts and Actions	142
Picker Views	146
Toolbars	152
Text	156
Web Views	164
Controls	170

Views

A view is the common name given to instances of **UIView**. You can think of a view as your application's canvas; in other words, if you are adding UI elements to your iPhone's interface, you are adding them to a view. All the UI elements discussed in this chapter are themselves subclasses of **UIView** and so inherit its properties and behavior.

The root level of your iPhone application interface consists of a single **UIWindow** to which you would typically add one or more views to work with, instead of using **UIWindow** directly.

Since **UIView** is a subclass of **UIResponder**, it can receive touch events. For most views, you'll receive only a single-touch event unless you set the **multipleTouchEnabled** property to **TRUE**. You can determine whether a view can receive touch events by modifying its **userInteractionEnabled** property. You can also force a view to be the only view to receive touch events by setting the **exclusiveTouch** property to **YES**. (For more information on working with touch events, see Chapter 7, "Touches, Shakes, and Orientation.")

You can also nest views within each other in what's known as the *view hierarchy*. Child views are known as *subviews*, and a view's parent is its *superview*.

Frames

Views are represented by a rectangular region of the screen called a *frame*. The frame specifies the *origin* (x, y) and *size* (width, height) of the view, in relation to its parent superview. The origin of the coordinate system for all views is the upper-left corner of the screen **(**.





A Child views (subviews) are nested inside their parent view (superview). A view's origin is at the top-left corner.



B Adding a subview to the view controller's main view.

000		🛅 UlTest – Debug	ger Conso	ole				0
				3			0	1
[Session st 2009-06-22 2009-06-22 2009-06-22 2009-06-22	arted at 200 11:47:14.153 11:47:14.154 11:47:14.155 11:47:14.155	9-06-22 11:47:12 UTDest[#199:20b] UTDest[#199:20b] UTDest[#199:20b] UTDest[#199:20b]	<pre>*1000.] frame: { bounds: frame: { bounds: frame: { bounds:</pre>	{10, {{0, {35, {10,	10}, 0}, { 10}, { 0}, {	{100, 100, (100, 100,	100}} 100}} 100}} 100}}	
UITest launche	d					(@ Succee	ded

Console output after moving the view. Notice that although the frame changes, the bounds remain the same.

Code Listing 4.1 Creating a new view.

To add a view to your application:

 Create a CGRect to represent the frame of the view, and pass it as the first parameter of the view's initWithFrame: method:

CGRect viewFrame = CGRectMake → (10,10,100,100);

UIView *myView = [[UIView alloc] → initWithFrame:viewFrame];

Here you are creating a view that is inset 10 pixels from the top left of its superview and that has a width and height of 100 pixels.

 Since the view is transparent by default, set its background color before adding it to the view controller's existing view B:

myView.backgroundColor =
→ [UIColor blueColor];

[[self view] addSubview:myView];

Code Listing 4.1 shows the completed code.

(II) To improve performance, set your UIView's opaque property to YES wherever possible.

Bounds

A view's *bounds* are similar to its frame, but the location and size are relative to the view's *own* coordinate system rather than those of its superview. In the previous example, the frame's origin is **{10,10}**, but the origin of its bounds is **{0,0}**. (The width and height for both the frame and the bounds are the same.)

The console output illustrates this **(**: After moving the view 25 pixels in the x direction (using the view's **center** property), the frame origin is now **(35,10)**, whereas the bounds origin remains at **(0,0)**.

Let's say you want to create a view so that it completely fills its superview. A common mistake is to use the frame of the superview.

If you tried to run this code in your application, you'd see a gap at the top of the subview **D**.

Recall that, in the project, the **UIWindow** is at the top level. The **UIWindow** has two subviews: the status bar and the main view 20 pixels below (2). The origin of the frame of the main view is actually **{0,20}**. (Remember, a view's frame is in relation to its superview's coordinate system.)

The solution to this problem is to use the *bounds* of the superview (Code Listing 4.2), which causes the view to correctly fill its superview.

You can use the NSStringFromCGRect()
function to convert a CGRect into an
NSString, making it useful for logging
CGRects to the console via NSLog().
Other useful functions when dealing with
CGRects are NSStringFromCGPoint()
and NSStringFromCGSize().



• Setting the frame incorrectly by using the frame of the superview. Notice the gap at the top.



• The enclosing **UIWindow** contains both the status bar and the view controller's view as subviews. Notice how the controller's view has an origin starting at **{0,20}** for its **frame**.

Code Listing 4.2 Initializing the view's frame with its superview's bounds.

```
- (void)viewDidLoad {
    UIView *myView = [[UIView alloc] initWithFrame:
        [self.view bounds]];
    myView.backgroundColor = [UIColor blueColor];
    [self.view addSubview:myView];
    [myView release];
}
```

Animation

Many properties of a view can be animated, including its **frame**, **bounds**, **backgroundColor**, **alpha** level, and more. You'll now look at some simple examples that illustrate additional view concepts.

To animate your view:

1. Retrieve the center of the view controller's main view:

```
CGPoint frameCenter = → self.view.center;
```

2. Create a view, set its background color, and, just as you did earlier, add it to the main view:

float width = 50.0;

float height = 50.0;

CGRect viewFrame = CGRectMake

- ightarrow (frameCenter.x-width,
- → frameCenter.y-height,width*2,
- ightarrow height*2);

UIView *myView = [[UIView alloc] → initWithFrame:viewFrame];

myView.backgroundColor =
→[UIColor blueColor];

[[self view] addSubview:myView];

Here you are positioning your view in the center of its superview and giving it a width and height of 50 pixels.

3. Set up an animation block:

[UIView beginAnimations:nil → context: NULL];

[UIView setAnimationDuration:1.0];

An animation block is a wrapper around a set of changes to animatable properties. In this example, the animation lasts for one second.

continues on next page

4. Resize the view:

viewFrame = CGRectInset(viewFrame, → -width, -height);

[myView setFrame:viewFrame];

The **CGRectInset()** function takes a source rectangle and then creates a smaller or larger rectangle with the same center point. In this example, a negative value for the width and height creates a larger rectangle.

5. Close the animation block:

[UIView commitAnimations];

This will cause all of the settings within the animation block to be applied.

6. Build and run the application.

You should see the view grow in size over a period of one second. **Code Listing 4.3** shows the completed code.

TP Try changing the setAnimationDuration: line to see how you can affect the speed of the animation.

(IIP) Try setting some other properties on the view within the animation block (such as backgroundColor) to see what effect they have.

Code Listing 4.3 Animating a view.

```
- (void)viewDidLoad {
    [super viewDidLoad];
   CGPoint frameCenter = self.view.center;
    float width = 50.0;
   float height = 50.0;
   CGRect viewFrame = CGRectMake(frameCenter.x-width,frameCenter.y-height, width*2, height*2);
   UIView *myView = [[UIView alloc] initWithFrame:viewFrame];
    myView.backgroundColor = [UIColor blueColor];
    [[self view] addSubview:myView];
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:1.0];
    viewFrame = CGRectInset(viewFrame, -width, -height);
    [myView setFrame:viewFrame];
    [UIView commitAnimations];
    [myView release];
}
```



Animating multiple views without using an autoresizing mask. Notice how the new subview ends up in the top-left corner of its superview.

Autosizing

When a view changes size or position, you often want any subviews contained within the view to change size or position in proportion to their containing superview. You can accomplish this by using a view's *autoresizing mask*. Now let's add a second subview inside the view you created in the previous exercise.

To add a subview:

 Create a CGRect for the subview's frame, again using the shortcut CGRectInset() function:

CGRect subViewFrame =

- → CGRectInset(myView.bounds,
- ightarrow width/2.0, height/2.0);

UIView *mySubview = →[[UIView alloc]

→ initWithFrame:subViewFrame];

mySubview.backgroundColor = →[UIColor yellowColor];

[myView addSubview:mySubview];

This time, the positive width and height values for the **CGRectInset** function make the new view smaller. To make them stand out, give it a different background color.

2. Build and run the application ①. The new subview starts off in the center of its superview, but then it remains "pinned" to its initial location as the animation progresses and ends up in the top-left corner.

continues on next page

Code Listing 4.4 shows this code updated to use an autoresizing mask. Notice how you set all four margins of the subview using the bitwise OR operator (the I symbol) between the constant values (Table 4.1). Notice also that even though the animation is specified on the superview, the subview still animates automatically **G**.

3. You can visually set the **autoresizingMask** property in the size pane of the Inspector window in Interface Builder **(H**).

Custom drawing

By default, the visual representation of a **UIView** is fairly boring. You can manipulate the size, background color, and alpha levels of the view, but not much else.

Luckily, it's relatively simple to create your own **UIView** subclasses where you can implement custom drawing behavior. To see how this might be done, you'll now learn how to create a **UIView** subclass with rounded corners.



G Using the autoresizing mask property, the subview remains in the center of its superview during an animation.



H Setting the autoresizing mask in Interface Builder.

Code Listing 4.4 Using an autoresizing mask.

```
- (void)viewDidLoad {
    [super viewDidLoad];
   CGPoint frameCenter = self.view.center;
   float width = 50.0:
   float height = 50.0;
   CGRect viewFrame = CGRectMake(frameCenter.x-width,frameCenter.y-height, width*2, height*2);
   UIView *myView = [[UIView alloc] initWithFrame:viewFrame];
   myView.backgroundColor = [UIColor blueColor];
    //create subview
   CGRect subViewFrame = CGRectInset(myView.bounds, width/2.0, height/2.0);
   UIView *mySubview = [[UIView alloc] initWithFrame:subViewFrame];
   mySubview.backgroundColor = [UIColor yellowColor];
    //set autoresizing mask
   mySubview.autoresizingMask = UIViewAutoresizingFlexibleLeftMargin
                                 UIViewAutoresizingFlexibleRightMargin
                                 UIViewAutoresizingFlexibleTopMargin
                                UIViewAutoresizingFlexibleBottomMargin;
    [myView addSubview:mySubview];
    [[self view] addSubview:myView];
    //animate resize
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:1.0];
    viewFrame = CGRectInset(viewFrame, -width, -height);
    [myView setFrame:viewFrame];
    [UIView commitAnimations];
    [mySubview release];
    [myView release];
}
```

TABLE 4.1 Available autoresizingMask values

Value	Description
UIViewAutoresizingNone	The view does not resize.
UIViewAutoresizingFlexibleLeftMargin	The view resizes by expanding or shrinking in the direction of the left margin.
UIViewAutoresizingFlexibleWidth	The view resizes by expanding or shrinking its width.
UIViewAutoresizingFlexibleRightMargin	The view resizes by expanding or shrinking in the direction of the right margin.
UIViewAutoresizingFlexibleTopMargin	The view resizes by expanding or shrinking in the direction of the top margin.
UIViewAutoresizingFlexibleHeight	The view resizes by expanding or shrinking its height.
UIViewAutoresizingFlexibleBottomMargin	The view resizes by expanding or shrinking in the direction of the bottom margin.

To create a custom rounded-corner view:

- In Xcode, select File > New File. Create a new Objective-C class, making sure that "Subclass of" is set to UIView ①. Save the file as roundedCornerView.
- 2. Open roundedCornerView.m, and modify your code to look like Code Listing 4.5.
- Open UITestViewController.m, and replace all instances of UIView with roundedCornerView. Don't forget to also import the header file for roundedCornerView.h at the top of the file. Code Listing 4.6 shows the updated code.
- Build and run your application to see the result with rounded corners for the views ①.

As you can see, custom drawing happens in the drawRect: method of roundedCornerView. You set a couple of variables here—one to determine the width of the line you will be drawing and another to determine the color.

 By setting the color to the superview's background color, you are essentially "erasing" any time you draw in the subview.

float lineWidth = 10.0;

UIColor *parentColor = [[self → superview] backgroundColor];

continues on page 122

IPhone QS coron I Corch Classs Code Signing Lesource Jser Interface	0bjective-C class 0bjective-C test utViewController subclass
Mac OS X AppleScript C and C++ Carbon Cocea Interface Builder SDK Pure Python Rulby Sync Services Other	Subclass of UNiew 5
	Description An Objective-C class which is a subclass of UFVew, wi an optional header file which includes the <urin <br="">UREL/h> header.</urin>



		and the second se	
			MA -
			п.
-III Carrier	> 1:15 PM		
and ourret			
		1000	
	6		
-			

() In the updated application, the views now have rounded corners.

Code Listing 4.5 The roundedCornerView class.

```
@implementation roundedCornerView
- (id)initWithFrame:(CGRect)frame {
    if (self = [super initWithFrame:frame])
    £
        self.opaque = TRUE;
    return self;
}
void CGContextStrokeCorners(CGContextRef ctx, CGRect rect) {
    int radius = 12;
    CGFloat xOrigin = rect.origin.x;
    CGFloat yOrigin = rect.origin.y;
    CGFloat xMiddle = CGRectGetMidX(rect);
    CGFloat yMiddle = CGRectGetMidY(rect);
    CGFloat width = rect.size.width;
    CGFloat height = rect.size.height;
    CGContextBeginPath(ctx);
    CGContextMoveToPoint(ctx, xOrigin, yMiddle);
    CGContextAddArcToPoint(ctx, xOrigin, yOrigin, xMiddle, yOrigin, radius);
    CGContextAddArcToPoint(ctx, width, yOrigin, width, yMiddle, radius);
    CGContextAddArcToPoint(ctx, width, height, xMiddle, height, radius);
CGContextAddArcToPoint(ctx, xOrigin, height, xOrigin, yMiddle, radius);
    CGContextClosePath(ctx);
    CGContextStrokePath(ctx);
}
- (void)drawRect:(CGRect)rect {
    float lineWidth = 10.0;
    UIColor *parentColor = [[self superview] backgroundColor];
    CGContextRef ctx = UIGraphicsGetCurrentContext();
    CGContextSetStrokeColorWithColor(ctx, parentColor.CGColor);
    CGContextSetLineWidth(ctx, lineWidth);
    //draw corners
    CGContextStrokeCorners(ctx,rect);
}
- (void)dealloc {
    [super dealloc];
}
Gend
```

6. Now you get a reference to the current graphics context and set the pen color and width.

A graphics context is a special type that represents the current drawing destination, in this case the custom view's contents.

a line around the outside of the view, rounding at each corner:

CGContextStrokeCorners(ctx,rect);

Code Listing 4.6 Replacing regular views with the custom class.

```
#import "UITestViewController.h"
#import "roundedCornerView.h"
@implementation UITestViewController
- (void)viewDidLoad {
    [super viewDidLoad];
   CGPoint frameCenter = self.view.center;
   float width = 50;
   float height = 50;
   CGRect viewFrame = CGRectMake(frameCenter.x-width, frameCenter.y-height,width*2, height*2);
   roundedCornerView *myView = [[roundedCornerView alloc] initWithFrame:viewFrame];
   myView.backgroundColor = [UIColor blueColor];
    [[self view] addSubview:myView];
   CGRect subViewFrame = CGRectInset(myView.bounds,width/2,height/2);
    roundedCornerView *mySubview = [[roundedCornerView alloc] initWithFrame:subViewFrame];
    mySubview.autoresizingMask = UIViewAutoresizingFlexibleHeight | UIViewAutoresizingFlexibleWidth;
    mySubview.backgroundColor = [UIColor yellowColor];
    [myView addSubview:mySubview];
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:1.0];
    viewFrame = CGRectInset(viewFrame, -width, -height);
    [myView setFrame:viewFrame];
    [UIView commitAnimations];
    [mySubview release];
    [myView release];
}
```

Transforms

You've already looked at resizing a view by increasing the width and height of its frame. Another way to perform the same task is by using a *transform*.

A transform maps the coordinates system of a view from one set of points to another. Transformations are applied to the *bounds* of a view. In addition to scaling, you can also rotate and move a view using transforms.

To resize your view using a scale transform:

Add the following code to your application:

CGAffineTransform scale = → CGAffineTransformMakeScale → (2.0,2.0);

myView.transform = scale;

This creates a scale transform, doubling both the width and the height of your view.

or

Transforms can also be used to move views by using a *translate* transform:

CGAffineTransform translate = → CGAffineTransformMakeTranslation → (50,50);

myView.transform = translate;

This would cause a view to move by 50 pixels along both the x- and y-axes.

or

Finally, you can apply a *rotation* transform to rotate your views:

CGAffineTransform rotate =

- ightarrow CGAffineTransformMakeRotation
- → (radiansForDegrees(180));

myView.transform = rotate;

Because rotations are specified in radians, you use a function to convert from degrees.

To apply both a rotation transform and a scale transform to your view:

1. Update the code to look like the following:

CGAffineTransform scale =

- → CGAffineTransformMakeScale
- → **(2.0,2.0);**

CGAffineTransform rotate =

- ightarrow CGAffineTransformMakeRotation
- → (radiansForDegrees(180));

CGAffineTransform myTransform =

- ightarrow CGAffineTransformConcat
- → (scale,rotate);

myView.transform = myTransform;

Note how you can combine transformations using the **CGAffineTransform Concat()** function.

Code Listing 4.7 shows the completed code.

2. Build and run your application \mathbf{K} .

Your view should rotate and scale at the same time.

TP You no longer need to set the autoresizingMask property of the subview because the transform is applied to the view and its subviews at the same time.

(IIP) You can return a view to its original state by setting its transform property to CGAffineTransformIdentity.



K The view both rotating and scaling.

Code Listing 4.7 Rotating and scaling the view.

```
CGFloat radiansForDegrees(CGFloat degrees)
{
    return (M_PI * degrees / 180.0);
3
- (void)viewDidLoad {
    [super viewDidLoad];
    CGPoint frameCenter = self.view.center;
    float width = 50;
    float height = 50:
    CGRect viewFrame = CGRectMake(frameCenter.x-width, frameCenter.y-height,width*2, height*2);
    roundedCornerView *myView = [[roundedCornerView alloc] initWithFrame:viewFrame];
    myView.backgroundColor = [UIColor blueColor];
    [[self view] addSubview:myView];
    CGRect subViewFrame = CGRectInset(myView.bounds,width/2,height/2);
    roundedCornerView *mySubview = [[roundedCornerView alloc] initWithFrame:subViewFrame];
    mySubview.backgroundColor = [UIColor yellowColor];
    [myView addSubview:mySubview];
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:1.0];
    CGAffineTransform scale = CGAffineTransformMakeScale(2.0,2.0);
    CGAffineTransform rotate = CGAffineTransformMakeRotation(radiansForDegrees(180));
    CGAffineTransform myTransform = CGAffineTransformConcat(scale,rotate);
    myView.transform = myTransform;
    [UIView commitAnimations];
    [mySubview release];
    [myView release];
3
```

Image Views

The **UIImageView** class extends **UIView** to provide support for displaying images. Its default initializer, **initWithImage:**, takes a **UIImage** as its only parameter (**Code Listing 4.8**):

```
UIImage *anImage = [UIImage

→ imageNamed:@"myImage.png"];
```

```
UIImageView *myImageView =
```

```
→[[UIImageView alloc]
```

```
\rightarrow initWithImage:anImage];
```

Note that **initWithImage:** automatically adjusts the frame of the new image view to match the width and height of the image assigned **(A)**.

If you resize the image view, you can see that the image automatically scales to fit $\ensuremath{\mathfrak{G}}$:

```
CGSize viewSize =

→ myImageView.bounds.size;

//shrink width 50%

viewSize.width = viewSize.width

→ * 0.5;

//keep height the same

viewSize.height = viewSize.height;

CGRect newFrame = CGRectMake

→ (0,0,viewSize.width,
```

```
→ viewSize.height);
```

[myImageView setFrame:newFrame];

Code Listing 4.8 Creating an image view.



A The image displaying a graphic.







Resizing the image view while maintaining its aspect ratio.

Code Listing 4.9 Animating over an array of images.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    NSArray *arrImages = [[NSArray alloc] initWithObjects:
                          [UIImage imageNamed:@"apple.png"],
                          [UIImage imageNamed:@"apple2.png"],
                          [UIImage imageNamed:@"apple3.png"],nil];
    CGRect viewFrame = CGRectMake(0,0,200,200);
    UIImageView *myImageView = [[UIImageView alloc] initWithFrame:viewFrame];
    [myImageView setAnimationImages:arrImages];
    [myImageView setAnimationRepeatCount:0];
    [myImageView setAnimationDuration:0.5];
    [self.view addSubview:myImageView];
    [myImageView startAnimating];
    [arrImages release];
    [myImageView release];
}
```

You can control scaling behavior by the **contentMode** property of **UIView**, which defaults to **UIViewContentModeScaleToFill**.

For example, to maintain the aspect ratio of the image, you would write this:

myImageView.contentMode = → UIViewContentModeScaleAspectFit;

In the resulting image, note that although the *image* itself is scaled, the *image view* still has the same bounds **()**. Any part of the bounds not rendered in the image will be transparent.

Animating images

UIImageView lets you animate over an array of images, which is handy for creating progress animations. **Code Listing 4.9** shows the code updated to animate over three images.

To animate over an image:

1. Create the image view, and set its frame:

CGRect viewFrame = CGRectMake → (0,0,200,200);

- UIImageView *myImageView =
- → [[UIImageView alloc]
- → initWithFrame:viewFrame];
- 2. Create and set the image array:

```
NSArray *arrImages =
→[[NSArray alloc] initWithObjects:
```

```
[UIImage imageNamed:
→ @"apple.png"],
```

[UIImage imageNamed: → @"apple2.png"],

[UIImage imageNamed: → @"apple3.png"],nil];

```
[myImageView
→ setAnimationImages:arrImages];
```

```
[arrImages release];
```

You can control the speed of the animation (in seconds) and number of times the animation is repeated. The default is
 making the animation loop indefinitely:

```
[myImageView
→ setAnimationDuration:0.5];
```

```
[myImageView setAnimation

→ RepeatCount:0];
```

4. To begin the animation, add the following:

[myImageView startAnimating];

 To stop the animation, you call stopAnimating.

W For simplicity, the previous examples use imageNamed: to create the images. Although convenient, this method creates autoreleased objects that can't be manually released in a lowmemory situation. So, it's usually wiser to use something like the initWithContentsOfFile: method and manually allocate/release your images.

Scrolling

Often your views will be larger than the visible area, and you need a way to scroll. For this, you use the **UIScrollView** class.

A scroll view acts as a container for a larger subview, allowing you to pan around the subview by touching the screen. Vertical and horizontal scroll bars indicate the position in the subview.

Code Listing 4.10 shows an example of using a scroll view.

To create a scroll view:

1. Set the frame as usual:

CGRect scrollFrame = CGRectMake \rightarrow (20,90,280,280);

UIScrollView *scrollView =

- → [[UIScrollView alloc]
- → initWithFrame:scrollFrame];
- **2.** Create an image view, assigning it an image that is larger than the scroll view:

UIImage *bigImage = [UIImage → imageNamed:@"appleLogo.jpg"];

UIImageView *largeImageView =

- \rightarrow [[UIImageView alloc]
- → initWithImage:bigImage];

continues on next page

Code Listing 4.10 Using a scroll view.

```
- (void)viewDidLoad {
  [super viewDidLoad];
  CGRect scrollFrame = CGRectMake(20,90,280,280);
  UIScrollView *scrollView = [[UIScrollView alloc] initWithFrame:scrollFrame];
  UIImage *bigImage = [UIImage imageNamed:@"appleLogo.jpg"];
  largeImageView = [[UIImageView alloc] initWithImage:bigImage];
  [scrollView addSubview:largeImageView];
  scrollView.contentSize = largeImageView.frame.size; //important!
  [self.view addSubview:scrollView];
  [scrollView release];
}
```
Add the image view to the scroll view, and set the contentSize property of the scroll view:

[scrollView addSubview: →largeImageView];

scrollView.contentSize =
→largeImageView.frame.size;

This is an important step: If you don't tell the scroll view how large its subview is, it won't know how to scroll at all.

4. Finally, add the scroll view to the main view:

[self.view addSubview:scrollView];

You'll now see the scroll view with horizontal and vertical scroll bars indicating the current position in the image view O. You can hide these scroll bars using the showsHorizontalScrollIndicator and showsVerticalScrollIndicator properties.

If you play around with the previous code, you'll notice that if you scroll quickly to the edge of the subview, the scroll view actually moves a little too far before springing back. This behavior is controlled by the bounce property. You can restrict bouncing to the x- or y-axis using the alwaysBounceHorizontal and alwaysBounceVertical properties, or you can disable it entirely by setting bounce to NO.

Zoom

You can also zoom in and out of an image using a scroll view. The **minimumZoom-**Scale and **maximumZoomScale** properties control the scale by which you can zoom in and out. By default, both of these properties are set to the same value (1.0), which disables zooming. You must implement one of the UIScrollViewDelegate methods to return the view that is being zoomed.



A Using a scroll view to pan around a large image.



B The page control indicating the total number of pages and the current page as a series of dots at the bottom of the iPhone's screen.

To enable zooming:

1. Add the UIScrollViewDelegate protocol in the controller.h file:

@interface UITestViewController : →UIViewController

- → <UIScrollViewDelegate>
- Update the scroll view code to allow you to zoom out by 1/2 and in by 2x:

scrollView.minimumZoomScale = 0.5;

```
scrollView.maximumZoomScale = 2.0;
```

scrollView.delegate = self;

You've also set the delegate to be the controller (**self**).

3. Implement the viewForZoomingInScroll View: delegate method, and return the image view. Code Listing 4.11 shows the updated code.

Paging

Scroll views support the *paging* of their content—the ability to add multiple subviews as "pages" and then scroll between them as you might turn the pages of a book. Adding a **UIPageControl** will provide a visual depiction of your current page **B**.

Code Listing 4.11 Adding zoom to the scroll view.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    CGRect scrollFrame = CGRectMake(20,90,280,280);
    UIScrollView *scrollView = [[UIScrollView alloc] initWithFrame:scrollFrame];
    scrollView.minimumZoomScale = 0.5;
    scrollView.maximumZoomScale = 2.0;
    scrollView.delegate = self;
    UIImage *bigImage = [UIImage imageNamed:@"appleLogo.jpg"];
    largeImageView = [[UIImageView alloc] initWithImage:bigImage];
    [scrollView addSubview:largeImageView];
    scrollView.contentSize = largeImageView.frame.size; //important!
    [self.view addSubview:scrollView];
    [scrollView release];
}
  (UIView *)viewForZoomingInScrollView:(UIScrollView *)scrollView
{
    return largeImageView;
```

To create a page control:

 Update the code to remove the image from the scroll view, and set some new properties:

```
float pageControlHeight = 18.0;
int pageCount = 3;
```

```
CGRect scrollViewRect =

→[self.view bounds];

scrollViewRect.size.height -=

→ pageControlHeight;
```

```
myScrollView =
    → [[UIScrollView alloc]
    → initWithFrame:scrollViewRect];
```

myScrollView.pagingEnabled = YES;

The **pagingEnabled** property turns paging on for the scroll view.

2. Since you have three pages, set the **contentView** of the scroll view to be three times wider than its frame. You'll also turn off the scroll view indicators:

```
myScrollView.delegate = self;
```

3. Set up the page control by creating a frame below the scroll view, and add a target to the page control so that when it is tapped, it will call the **changePage:** method:

CGRect pageViewRect = →[self.view bounds];

pageViewRect.origin.y =
→ scrollViewRect.size.height;

myPageControl = [[UIPageControl
→ alloc] initWithFrame:

→ pageViewRect];

myPageControl.backgroundColor =
→ [UIColor blackColor];

myPageControl.numberOfPages =
→ pageCount;

myPageControl.currentPage = 0;

- [myPageControl addTarget:self
- → action:@selector(changePage:)
- \rightarrow orControlEvents:UIControlEvent
- → ValueChanged];
- Call the createPages method by adding three UIViews side by side to the scroll view to represent the three pages.
- **5.** Set the **backgroundColor** property of the views.

In a real-world application, these would be more interesting! At this stage, your scroll view will actually work, but you need some more work to get the page control to reflect the current page.

6. Implement the scrollViewDidScroll: delegate method:

CGFloat pageWidth = sender.frame. → size.width;

int page = floor((sender.content
→ Offset.x - pageWidth / 2) /
→ pageWidth) + 1;

myPageControl.currentPage = page;

continues on next page

This simply does some math to calculate your current page during the scroll and then updates the page control accordingly.

7. Finally, implement the changePage: method called when the page control is tapped:

```
int page = myPageControl.
→ currentPage;
CGRect frame = myScrollView.frame;
frame.origin.x = frame.size.width
→ * page;
frame.origin.y = 0;
[myScrollView scrollRectToVisible:
→ frame animated:YES];
```

This scrolls the scroll view horizontally based on the page you have selected in the page control. **Code Listing 4.12** shows the completed code.

Code Listing 4.12 Implementing a page control.

```
UIScrollView *myScrollView;
UIPageControl *myPageControl;
@implementation UITestViewController
- (void)loadScrollViewWithPage:(UIView *)page
£
    int pageCount = [[myScrollView subviews] count];
    CGRect bounds = myScrollView.bounds;
    bounds.origin.x = bounds.size.width * pageCount;
    bounds.origin.y = 0;
    page.frame = bounds;
    [myScrollView addSubview:page];
}
- (void)createPages
{
    CGRect pageRect = myScrollView.frame;
    //create pages
    UIView *page1 = [[UIView alloc] initWithFrame:pageRect];
    page1.backgroundColor = [UIColor blueColor];
    UIView *page2 = [[UIView alloc] initWithFrame:pageRect];
    page2.backgroundColor = [UIColor redColor];
    UIView *page3 = [[UIView alloc] initWithFrame:pageRect];
    page3.backgroundColor = [UIColor greenColor];
```

code continues on next page

Code Listing 4.12 continued

```
//add to scrollview
    [self loadScrollViewWithPage:page1];
    [self loadScrollViewWithPage:page2];
    [self loadScrollViewWithPage:page3];
    //cleanun
    [page1 release];
    [page2 release];
    [page3 release];
}
- (void)viewDidLoad {
    [super viewDidLoad];
    float pageControlHeight = 18.0;
    int pageCount = 3;
    CGRect scrollViewRect = [self.view bounds];
    scrollViewRect.size.height -= pageControlHeight;
    //create scrollview
    myScrollView = [[UIScrollView alloc] initWithFrame:scrollViewRect];
    myScrollView.pagingEnabled = YES;
    myScrollView.contentSize = CGSizeMake(scrollViewRect.size.width * pageCount.1);
    myScrollView.showsHorizontalScrollIndicator = NO;
    myScrollView.showsVerticalScrollIndicator = NO:
    myScrollView.delegate = self;
    //create pageview
    CGRect pageViewRect = [self.view bounds];
    pageViewRect.size.height = pageControlHeight;
    pageViewRect.origin.y = scrollViewRect.size.height;
    myPageControl = [[UIPageControl alloc] initWithFrame:pageViewRect];
    myPageControl.backgroundColor = [UIColor blackColor];
    myPageControl.numberOfPages = pageCount;
    myPageControl.currentPage = 0:
    [myPageControl addTarget:self action:@selector(changePage:) forControlEvents:UIControlEventValueChanged];
    //create pages
    [self createPages];
    //add to main view
    [self.view addSubview:mvScrollView]:
    [self.view addSubview:myPageControl];
    //cleanup
    [myPageControl release];
    [myScrollView release];
3
  (void)scrollViewDidScroll:(UIScrollView *)sender
{
    CGFloat pageWidth = sender.frame.size.width;
    int page = floor((sender.contentOffset.x - pageWidth / 2) / pageWidth) + 1;
    myPageControl.currentPage = page;
}
  (void)changePage:(id)sender
{
    int page = myPageControl.currentPage;
    // update the scroll view to the appropriate page
    CGRect frame = myScrollView.frame;
    frame.origin.x = frame.size.width * page;
    frame.origin.y = 0;
    [myScrollView scrollRectToVisible:frame animated:YES];
3
```

Labels

Instances of the **UILabel** class display a read-only view that can contain one or more lines of text. For example, to create a simple label and set its **text**, **font**, **textColor**, and **backgroundColor** properties (**Code Listing 4.13**), use this:

```
myLabel.backgroundColor =
→[UIColor clearColor];
```

myLabel.textColor =

```
→ [UIColor redColor];
```

```
myLabel.font = [UIFont
→ systemFontOfSize: 18.0];
```

```
myLabel.text = @"Hello World!";
```

By default, a label is rendered as black text on a white background. You can also set a font by name:

```
myLabel.font = [UIFont fontWithName:

→ @"Verdana" size:18.0];
```

Table 4.2 shows the available fonts youcan use.

If you don't specify a font size, the label will automatically reduce the font to fit the text within the label's frame. You can control how small the font gets with the **minimumFontSize** property, and you can disable this behavior entirely with the **adjustsFontSizeToFitWidth** property.

To add a shadow to a label's text, you could write the following:

```
myLabel.shadowColor =
→[UIColor darkGrayColor];
```

myLabel.shadowOffset =
→ CGSizeMake(1.0,1.0);

The **shadowOffset** controls set how far on the x- and y-axes from the label's text the shadow is drawn. The default is **{0,-1}**.

Code Listing 4.13 Creating a label.

Family	Name		
American Typewriter	AmericanTypewriter, AmericanTypewriter-Bold		
AppleGothic	AppleGothic		
Arial	ArialMT, Arial-BoldMT, Arial-BoldItalicMT, Arial-ItalicMT		
Arial Hebrew	ArialHebrew, ArialHebrew-Bold		
Arial Rounded MT Bold	ArialRoundedMTBold		
Arial Unicode MS	ArialUnicodeMS		
Courier	Courier, Courier-BoldOblique, Courier-Oblique, Courier-Bold		
Courier New	CourierNewPS-BoldMT, CourierNewPS-ItalicMT, CourierNewPS-BoldItalicMT, CourierNewPSMT		
DB LCD Temp	DBLCDTempBlack		
Geeza Pro	GeezaPro-Bold, GeezaPro		
Georgia	Georgia-Bold, Georgia, Georgia-BoldItalic, Georgia-Italic		
Hiragino Kaku Gothic ProN	HiraKakuProN-W6, HiraKakuProN-W3		
Heiti J	STHeitiJ-Medium, STHeitiJ-Light		
Heiti K	STHeitiK-Medium, STHeitiK-Light		
Heiti SC	STHeitiSC-Medium, STHeitiSC-Light		
Heiti TC	STHeitiTC-Light, STHeitiTC-Medium		
Helvetica	Helvetica-Oblique, Helvetica-BoldOblique, Helvetica, Helvetica-Bold		
Helvetica Neue	HelveticaNeue, HelveticaNeue-Bold		
Marker Felt	MarkerFelt-Thin		
Times New Roman	TimesNewRomanPSMT, TimesNewRomanPS-BoldMT, TimesNewRomanPS-BoldItalicMT, TimesNewRomanPS-ItalicMT		
Thonburi	Thonburi-Bold, Thonburi		
Trebuchet MS	TrebuchetMS-Italic, TrebuchetMS, TrebuchetMS-BoldItalic, TrebuchetMS-Bold		
Verdana	Verdana-Bold, Verdana-BoldItalic, Verdana, Verdana-Italic		
Zapfino	Zapfino		

TABLE 4.2 Fonts available on the iPhone

The **textAlignment** property allows you to align the label text to the left (the default), center, or right.

The **lineBreakMode** property controls how a label wraps text that is too wide to fit within its frame. You can specify whether you want the text to be word or character wrapped, clipped, or truncated at the start, end, or middle of the text.

To display multiple lines of text in a label, use the **numberOfLines** property and the **\n** newline escape character:

myLabel.numberOfLines = 2;

myLabel.text = @"Hello World\nSecond → line";

The height of the label's **frame** property needs to be tall enough to accommodate the number of lines of text you specify, or the text will be wrapped using the value defined in the **lineBreakMode** property (**Code Listing 4.14**).

IIP Setting the numberOfLines property to 0 will make the label dynamically set the line count.

Code Listing 4.14 Setting various properties of a label.



A progress view at 33 percent completion.

Progress and Activity Indicators

When performing tasks that may take some time, you often need to provide some kind of visual feedback to your users. If you know how long the task will take to complete, you can use a progress indicator to show the user how much of the task has been performed and how much still has to run. If you are unable to determine the duration of the task, use a "busy" indicator (such as the beach ball or hourglass on OS X).

iOS provides classes for showing both progress and activity.

Indicating progress

When you want to show the progress of a task, use **UIProgressView**, a very simple class, consisting of only two properties.

You create a progress view and set its style using the **initWithProgressViewStyle**: method:

UIProgressView *myProgressView =

- → [[UIProgressView alloc]
- $ightarrow extsf{initWithProgressViewStyle:}$
- → UIProgressViewStyleDefault];

The indicator appears as a horizontal bar that fills from left to right to show completion **(A)**. This is controlled by the **progress** property, using a value between **0.0** (not started) and **1.0** (completed):

[myProgressView setProgress:0.33];

Although you set the frame of the progress view, the maximum height of a progress view is 9 pixels, so any larger value will be ignored. **Code Listing 4.15** shows an example of using **UIProgressView** with the progress updated in a timer to simulate a long-running task.

(IP) The other progress bar style, UIProgress ViewStyleBar, also uses a horizontal bar indicator but is more suitable for using in a toolbar (explained in the following section).

Showing activity

For tasks of an indeterminate duration, you can use the **UIActivityIndicatorView** class, represented by an animated "spinner" graphic **B**.



B An activity indicator view.

Code Listing 4.15 Updating the progress view.

```
NSTimer *timer;
@implementation UITestViewController
- (void)updateProgress:(NSTimer *)sender
{
    UIProgressView *progress = [sender userInfo];
    //have we completed?
    if (progress.progress == 1.0)
        [timer invalidate];
    else
        progress.progress += 0.05;
}
- (void)viewDidLoad {
    [super viewDidLoad];
    UIProgressView *myProgressView = [[UIProgressView alloc] initWithProgressViewStyle:UIProgressViewStyleDefault];
    CGRect progressFrame = CGRectMake(10,100,300,25);
    [myProgressView setFrame:progressFrame];
    [myProgressView setProgress:0.0];
    [self.view addSubview:myProgressView];
    [myProgressView release];
    //create timer
    timer = [[NSTimer scheduledTimerWithTimeInterval:0.1
                                              taraet:self
                                            selector:@selector(updateProgress:)
                                            userInfo:myProgressView
                                             repeats:YES] retain];
}
```

Similar to a progress view, you can create an activity indicator using the **initWithActivityIndicatorStyle:** method:

UIActivityIndicatorView

- → *myActivityView =
- → [[UIActivityIndicatorView alloc]
- → initWithActivityIndicatorStyle:
- → UIActivityIndicatorViewStyleWhite];

The default size of an activity view is a 21-pixel square. If you use the **UIActivity IndicatorViewStyleWhiteLarge** style, this increases to a 36-pixel square.

Unlike the progress view, however, the frame property controls both the height and the width of the view. For activity views larger than 36 pixels, it's best to use the larger style so the image won't become pixelated.

The activity view will initially be invisible. Calling the **startAnimating** method shows the activity view and causes the spinner graphic to animate:

[myActivityView startAnimating];

Calling **stopAnimating** will stop the spinner animation, but you need to remember to set the **hidesWhenStopped** property if you want the activity view to hide (**Code Listing 4.16**).

Code Listing 4.16 Creating an activity indicator view.

Alerts and Actions

Often in your applications you'll want to present a message to your users. Perhaps you want to alert them about an error or present them with options for a given action. As an iPhone developer, you handle these situations using *alert views* and *action sheets*.

Alerting users

To display an alert message, use the UIAlert View class. You define a title, message, and delegate, and then you configure buttons to be shown in the view.

To display an alert view:

- **1.** First, create a simple alert view \mathbf{A} :
 - **UIAlertView** *myAlert =
 - →[[UIAlertView alloc]
 - \rightarrow initWithTitle:@"title"
 - \rightarrow message:@"message"
 - \rightarrow delegate:nil
 - \rightarrow cancelButtonTitle:@"OK",
 - \rightarrow otherButtonTitles:nil]
 - \rightarrow [myAlert show];
- Using the otherButtonTitles property, you can create the same alert view with up to four additional buttons D:
 - UIAlertView *myAlert =
 - →[[UIAlertView alloc]
 - \rightarrow initWithTitle:@"title"
 - → message:@"message"
 - → delegate:nil
 - → cancelButtonTitle:@"OK"
 - \rightarrow otherButtonTitles:@"button1",
 - \rightarrow @"button2", @"button3",
 - ightarrow @"button4",nil];



A bare-bones alert view.



B An alert view with several buttons added.

Using the dismissWithClickedButton Index:animated: method, you can programmatically close an alert view without the user having to tap a button. This might be useful in a situation where you want to show an alert for a short time and then hide it automatically.

Code Listing 4.17 Display an alert view.

If you have only two buttons in your alert view, they will be displayed side by side. Otherwise, buttons are added from top to bottom, with the cancel button always being at the very bottom. If you don't need the message or title text, there is room for five buttons in addition to the cancel button.

 You can add buttons after creating your alert view by using the addButtonWith Title: method:

[myAlert addButtonWithTitle: →@"new button"];

 To determine which button is tapped, set the delegate, and implement the alertView:clickedButtonAtIndex: delegate method (Code Listing 4.17).

The **buttonIndex** parameter tells you which button was tapped, starting with the cancel button at index 0. Alert views close automatically when a button is tapped.

(IP) Another way to alert the user is by making the iPhone vibrate by calling the function AudioServicesPlayAlertSound (kSystemSoundID_Vibrate). You'll need to add the AudioToolbox framework to your project for this to work.

Confirming an action

When presenting the user with a number of options, you can use a **UIActionSheet**.

To create an action sheet:

- An action sheet is created in a similar way to an alert view ():
 - **UIActionSheet** *mySheet =
 - \rightarrow [[UIActionSheet alloc]
 - → initWithTitle:@"Do you really
 - → want to delete?" delegate:nil
 - → cancelButtonTitle:@"No"
 - → destructiveButtonTitle:@"Yes"
 - → otherButtonTitles:nil];

[mySheet showInView:self.view];

2. Define titles for three types of button.

The *cancel* button is generally used to dismiss the action sheet.

The *destructive* button acts as the confirmation of the action and is usually shown in red to indicate its importance.

The *other* buttons are similar to the alert view and allow you to add more buttons.

Setting any of these parameters to **nil** prevents the button type from showing.

 Set the delegate, and implement the actionSheet:clickedButtonAtIndex: method, which is called when a button is tapped.

You can compare the **buttonIndex** parameter to the action sheet's **cancelButtonIndex** and **destructiveButtonIndex** properties to determine which button was tapped.

Code Listing 4.18 shows the code updated with some of these options.

.atil Carrier 🙃 2:31 PM 🔛
Do you really want to delete?
Vec
Yes
No

C An action sheet is "pinned" to the bottom of the screen, and it contains only a title.

Code Listing 4.18 Adding more options to the action sheet.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    UIActionSheet *mySheet = [[UIActionSheet alloc] initWithTitle:@"Email Deletion Options"
                                                         delegate:self
                                               cancelButtonTitle:@"Cancel"
                                           destructiveButtonTitle:@"Delete Everything"
                                               otherButtonTitles:@"All Read Email", @"Spam Only",nil];
    mvSheet.actionSheetStyle = UIActionSheetStyleBlackOpaque:
    [mySheet showInView:self.view];
    [mySheet release];
3
- (void)actionSheet:(UIActionSheet *)actionSheet clickedButtonAtIndex:(NSInteger)buttonIndex {
    BOOL cancelClicked = actionSheet.cancelButtonIndex == buttonIndex:
    BOOL destructiveButtonClicked = actionSheet.destructiveButtonIndex == buttonIndex;
    NSLog(@"button with index %i clicked (cancel:%i, destructive:%i)",buttonIndex,cancelClicked,destructiveButtonClicked);
3
```

Action sheets vs. alert views

Action sheets are functionally similar to alert views, with a number of important differences:

- Action sheets are attached to a view. The code used in the previous exercise attaches the action sheet to the controller's main view.
- You can optionally show the alert sheet from a tab bar or a toolbar using the showFromTabBar: and showFromToolbar: methods.
- Action sheets do not have a message property; they have a single title property.
- You can change how the action sheet looks by using the actionSheetStyle property. In addition to the default style, you can give your action sheet a black transparent or opaque style. Setting the style to UIActionSheetStyleAutomatic will give your action sheet the same appearance as the bottom bar if one exists.

Picker Views

The **UIPickerView** class allows users to "spin" a wheel-type control to select one or more values. Each picker view consists of one or more *components* consisting of one or more *rows*. Each component can be spun independently of the others. For example, the picker view can be used to select a date value with three components in the control, representing the month, day, and year **(a)**.

The number of components and rows in a picker view is determined by its *datasource*, an object that adopts the **UIPickerViewDataSource** protocol. The display and selection of the picker view content is handled by the *delegate*, which adopts the **UIPickerViewDelegate** protocol (the datasource and the delegate can be the same object).

To create a simple picker view:

1. Add the protocol declarations to your interface definition:

@interface UITestViewController :

- \rightarrow UIViewController
- $ightarrow \mbox{curre}$,
vUIPickerViewDataSource,
- → UIPickerViewDelegate>
- 2. Create a picker view, and add it to the main view (Code Listing 4.19):

```
CGRect pickerFrame =
→ CGRectMake(0,120,0,0);
```

```
UIPickerView *myPicker =

→ [[UIPickerView alloc]

→ initWithFrame:pickerFrame];

myPicker.dataSource = self;

myPicker.delegate = self;
```

```
[self.view addSubview:myPicker];
```



A picker view being used to select a date.

Picker views are always 320 pixels by 216 pixels in size and cannot be resized.

- **3.** The **showsSelectionIndicator** property creates a translucent bar across the control to indicate the selected row.
- **4.** At a minimum, you need to implement two data source methods.

numberOfComponentsInPickerView:
returns the number of segments or
components in the picker view. In this
example, you want a single component,
so return the value 1.

pickerView:numberOfRowsInComponent:
returns the number of rows for each

component. Again, ignore the component parameter (since you have only a single component), and return the number of rows.

continues on next page

Code Listing 4.19 A bare-bones picker view implementation.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    CGRect pickerFrame = CGRectMake(0,120,0,0);
    UIPickerView *myPicker = [[UIPickerView alloc] initWithFrame:pickerFrame];
    mvPicker.dataSource = self;
    myPicker.delegate = self;
    myPicker.showsSelectionIndicator = YES;
    [self.view addSubview:myPicker];
    [myPicker release];
}
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView {
    return 1;
}
- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:(NSInteger)component {
    return 10;
}
- (NSString *)pickerView:(UIPickerView *)thePickerView titleForRow:(NSInteger)row forComponent:(NSInteger)component {
    return [NSString stringWithFormat:@"Row %i",row];
}
```

5. Implement the delegate pickerView: titleForRow:forComponent: method, returning an NSString representation of the current row D:

return [NSString stringWithFormat: →@"Row %i",row];

The picker view can display much more interesting data than this simple example. Components can be of different widths and, rather than just simple text, can actually have entire views embedded within them **(**.

To enhance the picker view:

- After calling the initComp1 and initComp2 methods to create some sample data, update the numberOfComponentsInPickerView: method to return two components (one for each of the sample arrays). Also, update the pickerView:numberof RowsInComponent: method to return the size of each array:
 - if (component == 0)

return [comp1 count];

else

return [comp2 count];

The arrays here contain different numbers of elements; in other words, components do not need the same number of rows.

2. Define a new delegate method, picker View:widthFormComponent:, and set the widths of the components to different values:

```
if (component == 0)
```

return 100.0;

```
else
```

return 200.0;



B The picker view shows the sample data.



(6) The updated picker view. Not only do the two components display different content, but they also have different widths and numbers of items.

- Implement the pickerView:viewForRow: forComponent:reusingView: delegate, returning either an image view or a label. This method allows you to embed almost any view subclass in a picker view component.
- Finally, in the pickerView:didSelectRow: inComponent: delegate, log the selected row and component to the console.

When you spin the picker view, this method isn't fired until the scrolling animation ends. **Code Listing 4.20** shows the updated code.

Code Listing 4.20 The updated picker view.

```
NSMutableArray *comp1;
NSMutableArray *comp2;
@implementation UITestViewController
-(void)initComp1
{
    comp1 = [[NSMutableArray alloc] init];
    UIImageView *imgView:
    imgView = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"ca.png"]];
    [comp1 addObject:imgView];
    [imgView release];
    imgView = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"gb.png"]];
    [comp1 addObject:imgView];
    [imgView release];
    imgView = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"us.png"]];
    [comp1 addObject:imgView];
    [imgView release];
}
-(void)initComp2
ł
    comp2 = [[NSMutableArray alloc] init];
UILabel *lbl;
    lbl = [[UILabel alloc] initWithFrame:CGRectMake(0,0,100,44)];
    lbl.backgroundColor = [UIColor clearColor];
    lbl.text = @"Red";
    [comp2 addObject:lbl];
    [lbl release];
    lbl = [[UILabel alloc] initWithFrame:CGRectMake(0,0,100,44)];
    lbl.backgroundColor = [UIColor clearColor];
    lbl.text = @"Blue";
    [comp2 addObject:lbl];
    [lbl release];
3
```

code continues on next page

Code Listing 4.20 continued

```
- (void)viewDidLoad {
    [super viewDidLoad];
    //create some sample data
    [self initComp1];
    [self initComp2];
    CGRect pickerFrame = CGRectMake(0,120,0,0);
    UIPickerView *myPicker = [[UIPickerView alloc] initWithFrame:pickerFrame];
    myPicker.dataSource = self:
    myPicker.delegate = self;
    myPicker.showsSelectionIndicator = YES;
    [self.view addSubview:myPicker];
    [myPicker release];
}
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView {
    return 2;
}
- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:(NSInteger)component {
    if (component == 0)
        return [comp1 count];
    else
        return [comp2 count];
}
- (CGFloat)pickerView:(UIPickerView *)pickerView widthForComponent:(NSInteger)component
{
    if (component == 0)
        return 100.0;
    else
        return 200.0;
}
- (UIView *)pickerView:(UIPickerView *)pickerView
            viewForRow: (NSInteger)row
          forComponent: (NSInteger )component
          reusingView:(UIView *)view {
    if (component == 0)
        return [comp1 objectAtIndex:row];
    else
        return [comp2 objectAtIndex:row];
}
- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row inComponent:(NSInteger)component {
    NSLog(@"row: %i, component:%i",row,component);
}
```

Picking dates and times

iOS also has a special version of a picker, **UIDatePicker**, geared toward picking dates as well as times. The **datePickerMode** property determines the style of the picker **D**.

Since the date picker is localized, it will automatically display dates and times in the format of the device locale. You can, however, override these settings to display dates and times for other locales.

You can set properties for start and end dates (for the date-style pickers) and for minute and countdown values (for the time-style pickers).

UIDatePicker is not actually a subclass of **UIPickerView**. It is a **UIControl** subclass that has a custom **UIPickerView** as a subview. This means that you use the target-action mechanism to manage the selection of values. As with other controls, you set the action:



D A date picker with the default style of UIDatePickerModeDateAndTime lets you pick both the date and the time.

[myPicker addTarget:self action:@selector(pickerChanged:) forControlEvents: → UIControlEventValueChanged];

The date picker creates a **UIControlEventValueChanged** event when a date or time is selected (Code Listing 4.21).

Code Listing 4.21 Implementing a date picker.

Toolbars

You can create toolbars in iPhone applications using the **UIToolbar** class. A toolbar usually spans the entire width of the display and is aligned to either the top or the bottom of the screen **(A)**.

To create a toolbar:

 As with many other views, you can create a toolbar with the initWithFrame: method. Use the size of the main view to calculate the y position of the toolbar. This is important since you may not know the orientation of the iPhone and want the toolbar to sit at the bottom of the screen.

```
CGSize viewSize =
→ self.view.frame.size;
```

float toolbarHeight = 44.0;

```
CGRect toolbarFrame = CGRectMake

→ (0,viewSize.height-toolbarHeight,

→ viewSize.width,toolbarHeight);
```

UIToolbar *myToolbar =

- → [[UIToolbar alloc] initWithFrame:
 → toolbarFrame];
- 2. Set the autoresizingMask property of the toolbar to ensure that it stays in the same position (in this case, aligned to the bottom of the screen) even if the user rotates their iPhone.

```
myToolbar.autoresizingMask =
```

- $\rightarrow \textbf{UIViewAutoresizingFlexible}$
- \rightarrow Width | UIViewAutoresizing
- → FlexibleLeftMargin | UIView
- → AutoresizingFlexibleRight
- → Margin | UIViewAutoresizing
- → FlexibleTopMargin;



A Most of the controls sit on the toolbar in Safari.



A toolbar showing the three toolbar item styles for the initWithTitle:style:target:action: method.

Save 🕨 🔍 🗙 🕂 🗐	

C Some of the available system item styles.

 You can change the color and translucency of the toolbar using the tintColor and translucent properties:

myToolbar.tintColor = →[UIColor redColor]; myToolbar.translucent = YES;

Toolbar items

Buttons you add to a toolbar are known as toolbar items and are created using the **UIBarButtonItem** class. Several types of buttons are available, and you can create them in several ways:

- The simplest way to create a button with some title text is by using the initWithTitle:style:target:action: method ^(B). Use the target and action parameters to indicate which method to call when the button is pressed.
- Similarly, the initWithImage:style: target:action: method lets you create a button with an image instead of text. The button will automatically resize its width to that of the image.
- You can create a button from your own custom UIView subclass using the initWithCustomView: method. However, you must set the target and action properties manually.
- The final way is to use the initWithBar ButtonSystemItem:target:action: method.

iOS offers a set of predefined buttons, known as *system items*, to ensure your application adheres to the iPhone interface guidelines. Use them whenever possible.

There are system items for play, pause, and stop buttons, as well as for search, trash, and camera **(**). For a complete list of system items available, refer to the **UIBarButtonSystemItem** type in the developer documentation. You will often use two particular system item types: **UIBarButtonSystemItemFlexibleSpace** and **UIBarButtonSystemItemFixedSpace**. Both are not visible and represent spaces on a toolbar. The flexible-space item lets you force a button to the other side of the toolbar, while the fixed-space item simply lets you add a space between buttons. You can set the **width** property of a fixed-space item to determine how wide you want the space to be.

Once you've created these buttons, add them to an NSArray and then use the setItems: method of the UIToolbar to add them to the toolbar itself. The optional animated: parameter allows you to have buttons fade in as they are added to the toolbar.

Code Listing 4.22 shows the updated code, with buttons of various types and a flexible-space item being used to push a button to the right side of the toolbar. If you try rotating the phone, you will notice that the toolbar and buttons correctly align themselves regardless of orientation **①**.



D The toolbar has correctly sized itself with the iPhone in landscape mode. The trash toolbar item is aligned to the right.

Code Listing 4.22 Creating several different types of toolbar items.

```
- (void)buttonClick:(id)sender {
    NSLog(@"you clicked button: %@",[sender title]);
3
- (void)viewDidLoad {
    [super viewDidLoad];
    CGSize viewSize = self.view.frame.size;
    float toolbarHeight = 44.0;
    CGRect toolbarFrame = CGRectMake(0.viewSize.height-toolbarHeight.viewSize.width.toolbarHeight):
    UIToolbar *myToolbar = [[UIToolbar alloc] initWithFrame:toolbarFrame];
    myToolbar.autoresizingMask = UIViewAutoresizingFlexibleWidth
                                 UIViewAutoresizingFlexibleLeftMargin
                                 UIViewAutoresizinaFlexibleRightMargin
                                 UIViewAutoresizingFlexibleTopMargin;
    UIBarButtonItem *button1 = [[UIBarButtonItem alloc] initWithTitle:@"button 1"
                                                                style:UIBarButtonItemStylePlain target:self
                                                               action:@selector(buttonClick:)];
    UIBarButtonItem *button2 = [[UIBarButtonItem alloc] initWithTitle:@"button 2"
                                                                style:UIBarButtonItemStyleBordered
                                                               target:self
                                                               action:@selector(buttonClick:)];
    UIBarButtonItem *button3 = [[UIBarButtonItem alloc] initWithImage:[UIImage imageNamed:@"apple_icon.png"]
                                                                style:UIBarButtonItemStyleBordered
                                                               taraet:self
                                                               action:@selector(buttonClick:)];
    UIBarButtonItem *flexButton = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemFlexibleSpace
                                                                                target:nil
                                                                                action:nil];
    UIBarButtonItem *trashButton = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemTrash
                                                                                 target:self
                                                                                 action:@selector(buttonClick:)];
    NSArray *buttons = [[NSArray alloc] initWithObjects:button1,button2,button3, flexButton,trashButton,nil];
    //cleanup
    [button1 release];
    [button2 release];
    [button3 release];
    [flexButton release];
    [trashButton release];
    [myToolbar setItems:buttons animated:NO];
    [buttons release];
    [self.view addSubview:myToolbar];
    [myToolbar release];
3
```

Text

For entering text into your applications, iOS provides two classes, **UIText Field** and **UITextView**. Both allow the user to enter and edit text using an onscreen keyboard and support features such as cut/copy and paste, spell check, and more, but the two classes function differently.

To create a text field:

- You can use the UITextField class to enter small amounts of text, such as user names, passwords, or search terms. This field is limited to a single line of text.
- As with most other views, you use the initWithFrame: method to create them:

CGRect textRect = CGRectMake → (10,10,300,20);

UITextField *myTextField =

→[[UITextField alloc]

→ initWithFrame:textRect];

```
myTextField.backgroundColor =
→[UIColor whiteColor];
```

This also sets the background color of the text field; otherwise, it's transparent by default. Text fields also don't have a border by default.

3. Use the **borderStyle** property to choose from four different styles **(A**).

The **UITextBorderStyleRoundedRect** style has a white background and will ignore the **backgroundColor** property. If you set a custom **UIImage** as the **background**, the **borderStyle** property will be ignored.

4. You can set the text font, color, and alignment to apply to the entire text field.

Text fields do not support the styling of individual text elements.



A Border styles available for text fields.

Clicking the clear button removes any text in the text field.



B Press the clear button to remove any text in the text field.





C Two of the keyboards available by setting the **keyboardType** property of a text field.

5. You can set your text field to automatically resize the font to accommodate larger text:

myTextField.font = → [UIFont systemFontOfSize:22.0];

myTextField.adjustsFontSizeTo
→ FitWidth = YES;

myTextField.minimumFontSize = 2.0;

This example sets the initial font size as 22 and then tells the text field to automatically shrink the font to a minimum size of 2 if the text is wider than the text field's bounds.

6. Setting the clearsOnBeginEditing property to YES will clear any existing text when you first touch the control:

myTextField.clearsOnBeginEditing = → YES;

The **clearButtonMode** property adds a small button to the end of the text field, letting you clear the text at any time **(B)**. You can determine when this button is shown, such as only when editing the text.

To use keyboards:

- **1.** Tap in the text field to open a keyboard from the bottom of the screen.
- You can choose from a number of keyboard styles (), each designed for particular situations such as entering numbers or using a web browser.
- **3.** Set the style with the **keyboardType** property.

By default, the keyboard will automatically suggest words as you type.

continues on next page

- To disable this function, set the autocorrectionType property to UITextAutocorrectTypeNo.
- **5.** Set the **autocaptializationType** property to determine whether the keyboard capitalizes your typing by word, sentence, or even all characters.
- You can change the text on the Return key via the returnKeyType property.
- 7. Use the enablesReturnKeyAutomatically property to determine whether the Return key is enabled even if you haven't entered any text into the text field.

In Code Listing 4.23, the secureTextEntry property is set to YES, which is useful for text fields that contain passwords or other sensitive information. As you enter text, you will see only the last letter typed.

 You may have noticed that the keyboard doesn't disappear when you press the Return key. To hide the text field, you must implement the textFieldShould Return: delegate and tell the text field to resign its first responder status:

[textField resignFirstResponder];

return YES;

Code Listing 4.23 Creating a secure text field.

```
- (void)viewDidLoad {
    CGRect textRect = CGRectMake(10,10,300,31);
   UITextField *myField = [[UITextField alloc]
                                initWithFrame:textRect];
   myField.borderStyle = UITextBorderStyleRoundedRect;
   myField.font = [UIFont systemFontOfSize:22.0];
   myField.adjustsFontSizeToFitWidth = YES;
   myField.minimumFontSize = 2.0:
   myField.clearButtonMode = UITextFieldViewModeWhileEditing;
   myField.keyboardType = UIKeyboardTypeDefault;
   myField.autocorrectionType = UITextAutocorrectionTypeNo;
   myField.autocapitalizationType = UITextAutocapitalizationTypeNone;
   myField.returnKeyType = UIReturnKeyDone;
   myField.secureTextEntry = YES;
    [self.view addSubview:myField];
    [myField release];
}
```

 Similarly, you can make the keyboard appear automatically when the view is loaded by setting the first responder status in the viewDidLoad: method:

[myTextField → becomeFirstResponder];

 To prevent the keyboard from showing at all, which is useful if you are implementing your own custom keyboard, return NO from the textFieldShould BeginEditing: delegate method.

TP For a complete list of keyboard options, refer to the "UITextInputTraits Protocol" section of the developer documentation.

Restricting content

You can also use the delegate methods to control the text being entered into the text field. The **textField:shouldChange CharactersInRange:replacementString:** delegate is called whenever the text is changed. You could, for example, use this method to restrict the number of characters entered. **Code Listing 4.24** shows a text field that allows a maximum of ten characters.

Code Listing 4.24 Limiting the contents of a text field to ten characters.

You should check the length of the replacement rather than just looking at the length of the text in the text field, since the text field's contents can be altered via copy and paste as well as by using the keyboard.

For the same reason, simply changing the keyboard type to numeric does not guarantee that a user will enter only numeric values (since a user could paste non-numeric values into the field). **Code Listing 4.25** shows the same delegate method, this time restricting the text field to allow numeric values only.

Text views

The **UITextView** class allows for multiline editable text. Although similar to text fields, text views feature a number of important differences.

Text views don't have any support for automatically reducing the font size like text fields have. Also, they don't have any support for clearing the text other than through programmatically setting the **text** property. There is also no support for secure text entry.

As with text fields, text views also apply the same text style to the entire text. Apple recommends using a **UIWebView** (see the "Web Views" section) if you require multiple styles in your text. **Code Listing 4.25** Restricting the contents of a text field to numeric values.



D A text view with an active data detector.

Data detectors

Text views can analyze their contents and convert any links or phone numbers into tappable links by using a capability known as *data detectors*. Tapping the link will either launch the browser or call the phone number.

Two data detector types are available: **UIData DetectorTypePhoneNumber** for phone numbers and **UIDataDetectorTypeLink** for Web http: links. To enable both, set the data DetectorTypes property:

myTextView.dataDetectorTypes = → UIDataDetectorTypeAll;

There's one caveat with data detectors: The default behavior of text views is to show the keyboard when tapped, so you can't tap the link of a data detector. For data detectors to work, you must set the **editable** property of the text view to **NO**. In **①**, the URL is underlined just as it would be in a web browser. Tapping it will launch the Safari application.

Hiding the keyboard

A text view's keyboard behaves the same as a text field, with one important difference: Since a text view supports multiline editing, pressing the Return button on the keyboard will insert a carriage return instead of calling a delegate method. Just as with the text field, resign the text view's first responder status to hide the keyboard when you have finished editing the text. This is often done as an action within another control.

Scrolling the interface

You may have noticed that since the iPhone's keyboards are very large, they take up a lot of the screen and can overlap other controls when shown. It would be handy if your interface moved up when the keyboard appeared and then moved back down once it disappeared.

You can make that happen by placing the controls inside a **UIScrollView**. When the keyboard appears, you simply scroll everything up, scrolling back down when the keyboard hides.

To scroll the interface in response to the keyboard:

1. Create and add a scroll view, making it the full size of the main view:

CGRect viewRect = →[self.view bounds];

```
myScrollView = [[UIScrollView
→ alloc] initWithFrame:viewRect];
```

myScrollView.contentSize =
→ viewRect.size;

[self.view → addSubview:myScrollView];

- 2. Add the controls to the scroll view instead of the main view (since you want them to scroll).
- 3. Implement the textViewDidBeginEditing: delegate method, which is called when the keyboard is shown.

Here you need to calculate both the bottom of the text view and the top of the keyboard and then tell the scroll view to scroll the difference. You must also look at the **orientation** property of the iPhone because the keyboard will have a different height in portrait mode than in landscape mode.

 Implement the textViewDidEndEditing: delegate so that when the keyboard is hidden, you scroll the text view to its original position. Code Listing 4.26 shows the completed code.

Code Listing 4.26 Scrolling an interface in response to a keyboard.

```
UIScrollView *mvScrollView:
UITextView *myTextView;
@implementation UITestViewController
-(void)buttonClick:(id)sender
{
    [myTextView resignFirstResponder];
}
- (void)viewDidLoad {
    CGRect viewRect = [self.view bounds];
    myScrollView = [[UIScrollView alloc] initWithFrame:viewRect];
    mvScrollView.contentSize = viewRect.size:
    [self.view addSubview:myScrollView];
    CGRect buttonFrame = CGRectMake(10,10,60,32);
    UIButton *keyboardToggle = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [keyboardToggle setTitle:@"hide" forState:UIControlStateNormal];
    [keyboardToggle addTarget:self action:@selector(buttonClick:) forControlEvents:UIControlEventTouchUpInside];
    keyboardToggle.frame = buttonFrame;
    [myScrollView addSubview:keyboardToggle];
    CGRect textRect = CGRectMake(10.60.300.200);
    myTextView = [[UITextView alloc] initWithFrame:textRect];
    myTextView.font = [UIFont systemFontOfSize:22.0];
    myTextView.keyboardType = UIKeyboardTypeDefault;
    myTextView.returnKeyType = UIReturnKeyGo;
    myTextView.delegate = self;
    [myScrollView addSubview:myTextView];
    [myTextView release];
    [myScrollView release];
}
- (void)textViewDidBeginEditing:(UITextView *)textView {
    float keyboardHeight;
    if ([UIDevice currentDevice].orientation == UIDeviceOrientationPortrait | UIDeviceOrientationPortraitUpsideDown)
        keyboardHeight = 216.0;
    else
        keyboardHeight = 162.0;
    CGRect textViewRect = textView.frame:
    float textViewBottom = textViewRect.origin.y + textViewRect.size.height;
    CGRect viewRect = [myScrollView bounds];
    float keyboardTop = viewRect.size.height-keyboardHeight;
    float scrollOffset = fabs(textViewBottom - keyboardTop);
    [myScrollView setContentOffset:CGPointMake(0, scrollOffset) animated:YES];
}
- (void)textViewDidEndEditing:(UITextView *)textView {
    [myScrollView setContentOffset:CGPointMake(0, 0) animated:YES];
}
```

Web Views

Just as with iPhone's native Safari application, you can display web-based content in your own applications by using the **UIWebView** class. (In fact, Safari on the iPhone uses a **UIWebView** for display.)

Web views provide touch-based control for zooming in and out of pages, panning, and scrolling. Tapping links can load pages, and tapping in text controls will open a keyboard for data entry.

To display a web page in your application:

 Just as with other views, you can add web views to your interface in the usual way:

```
CGRect webRect = CGRectMake
→ (10,10,300,400);
```

UIWebView *myWebView =

- →[[UIWebView alloc]
- → initWithFrame:webRect];

myWebView.scalesPageToFit = YES;

The **scalesPageToFit** property ensures that larger pages are zoomed out or in enough to fit correctly in the current frame as well as letting you zoom in and out in response to pinch gestures.

.ali Carr	ier 🗇	5:35	PM		
Web	Images	Local	News	more	7
_	0	GO	og	le ⁻	
	(Googl	e Search		
	<u>Sign in</u>	- <u>Pre</u>	ference	<u>s</u> - <u>He</u>	lp
1	View Go	ogle in	: Mobil	e <u>Clas</u>	sic

A web view displaying the Google homepage.

 Use the loadRequest: method to load content into the web view, which takes an NSURLRequest object as its only parameter:

NSURL *url = [NSURL URLWithString: →@"http://www.google.com"];

NSURLRequest *request = → [NSURLRequest requestWithURL:url];

[myWebView loadRequest:request];

[self.view addSubview:myWebView];

This would load the Google homepage (A).

 If your page is taking a long time or you want to cancel loading, use the stopLoading method. You can also check the loading property to make sure the page is actually in the process of loading.

What's the status?

Web views provide four optional delegate methods that will notify you about changes in the status of loading a web page:

webView:shouldStartLoadWithRequest:navigationType: is sent before the web view begins to load the content and is a handy place to handle navigation within your web views (see the "Loading local content and handling hyperlinks" section).

webViewDidStartLoad: is sent when your web page starts loading and is a good place to show a progress indicator.

webViewDidFinishLoad: is sent when the web view finishes loading a page and is a good place for you to stop a progress indicator. This will not be sent if the page fails to load for any reason.

webView:didFailLoadWithError: is sent if an error occurs in loading the web page.
If you are building a web browser-type interface with Forward and Backward buttons, you can use the **canGoForward** and **canGoBackward** properties to determine whether your buttons should be enabled, and you can use the **goForward** and **goBackward** methods to navigate through the web view's page history.

Although there is no direct access to the page history, you can easily maintain history via the delegate methods mentioned in the "What's the status?" sidebar. **Code Listing 4.27** shows the code updated to include an activity indicator when the page is loading.

Code Listing 4.27 Implementing a web view.

```
UIActivitvIndicatorView *activity:
@implementation UITestViewController
- (void)viewDidLoad {
    [super viewDidLoad];
    [self.view setBackgroundColor:[UIColor blackColor]];
    CGRect webRect = CGRectMake(10,10,300,380);
    UIWebView *myWebView = [[UIWebView alloc] initWithFrame:webRect];
    myWebView.scalesPageToFit = YES;
    mvWebView.delegate = self:
    NSURL *url = [NSURL URLWithString:@"http://www.google.com"];
    NSURLRequest *request = [NSURLRequest requestWithURL:url];
    [myWebView loadRequest:request];
    [self.view addSubview:myWebView];
    activity = [[UIActivityIndicatorView alloc] initWithActivityIndicatorStyle:UIActivityIndicatorViewStyleWhiteLarge];
    [activity setCenter:CGPointMake(160,420)];
    [self.view addSubview:activity];
    [myWebView release];
}
  (void)webViewDidStartLoad:(UIWebView *)webView
{
    [activity startAnimating];
}
  (void)webViewDidFinishLoad:(UIWebView *)webView
{
    [activity stopAnimating];
    [webView stringByEvaluatingJavaScriptFromString:@"alert('Finished Loading!');"];
}
- (void)webView:(UIWebView *)webView didFailLoadWithError:(NSError *)error
{
    [activity stopAnimating];
    NSLog(@"Error: %@",error);
}
```

Running JavaScript

You can execute JavaScript in a web view by using the **stringByEvaluatingJavaScript FromString:** method. For example, if you wanted to open an alert dialog box in your web view, write the following:

[webView stringByEvaluating → JavaScript FromString:@"alert → ('Hello World!);"];

This lets you manipulate your web page's style sheet or DOM or even call existing JavaScript functions defined within the page itself. For example, to change the background color of your web page, you could write the following:

[webView stringByEvaluating → JavaScript FromString: → @"document.bgColor= \"#000000\";"];

To call the JavaScript function **myFunction** defined in the web page, you could use this:

[webView stringByEvaluating → JavaScript FromString: → @"myFunction();"];

For performance reasons, any JavaScript you call must execute fully within ten seconds and must be less than 10MB in size.

Loading local content and handling hyperlinks

You can also use web views to display *local* content, such as an .html file that ships in your application bundle. This makes web views handy for displaying content that mixes graphics with text or requires multiple text styles. (Remember that **UILabel**s and **UITextView**s are restricted to only a single style per control.)

To load content from a local file:

1. Use the **loadHTMLString:baseURL:** method to load content contained in the resources folder of the application bundle:

myWebView.scalesPageToFit = NO;

NSString *htmlPath =

- → [[NSBundle mainBundle]
- → pathForResource:@"myPage"
- → ofType:@"html"];

NSString *htmlContent =

- \rightarrow [NSString stringWithContents
- → OfFile:htmlPath encoding:
- → NSUTF8StringEncoding error:nil];

[myWebView loadHTMLString: → htmlContent baseURL:nil];

This time, **scalePageToFit** has been set to **NO** because you don't want the user to be able to zoom in and out of the web view as if it were a web page **B**.

 To prevent a user from tapping any hyperlinks in the document, you can set userInteractionEnabled to NO for the web view. This also disables the ability to scroll content that may be longer than the control can fit on the screen at once.



B A web view displaying some local content, including a hyperlink to another page.

or

To disable links entirely, return **NO** from the webView:shouldStartLoadWith Request:navigationType: delegate method.

 To open a link in the native Safari application, you could write the following in the webView:shouldStartLoadWithRequest: navigationType: delegate method (Code Listing 4.28):

NSURL *pageURL = [request URL];

- if (([[pageURL scheme]
- → isEqualToString: @"http"]) &&
- → (navigationType == UIWebView
- → NavigationTypeLinkClicked))
- [[UIApplication sharedApplication] → openURL:pageURL];

return NO;

Here you are trapping only **http:** links. Other link types, such as **https:**, would have no effect.

When implementing this type of functionality, it's common to warn the user they are navigating away from your application and have them confirm the action. You can do this with the UIAlertView described in the exercise, "To display an alert view," earlier in this chapter.

Code Listing 4.28 Capturing clicks on a web view.

```
- (void)viewDidLoad {
  [super viewDidLoad]:
  CGRect webRect = CGRectMake(10,10,300,400);
 UIWebView *myWebView = [[UIWebView alloc] initWithFrame:webRect];
  myWebView.delegate = self;
  myWebView.scalesPageToFit = N0;
  NSString *htmlPath = [[NSBundle mainBundle] pathForResource:@"myPage" ofType:@"html"];
  NSString *htmlContent = [NSString stringWithContentsOfFile:htmlPath encoding:NSUTF8StringEncoding error:nil];
  [myWebView loadHTMLString:htmlContent baseURL:nil];
  [self.view addSubview:myWebView];
  [myWebView release];
}
- (BOOL)webView:(UIWebView *)webView
shouldStartLoadWithRequest:(NSURLRequest *)request
navigationType:(UIWebViewNavigationType)navigationType {
 NSURL *pageURL = [request URL];
 if ( ([[pageURL scheme] isEqualToString: @"http"]) && (navigationType == UIWebViewNavigationTypeLinkClicked ))
 £
    [[UIApplication sharedApplication] openURL:pageURL];
    return NO:
 }
 return YES:
3
```

Controls

Almost all the drawing functionality you've learned about so far also applies to controls. Most controls inherit their class from **UIControl**, and **UIControl** is a subclass of **UIView** (3); this is how controls know how to draw themselves.

You'll never actually create instances of **UIControl** directly the way you do with **UIView. UIControl** is simply used to define a common set of functionality and behavior for its subclasses.

As mentioned at the beginning of this chapter, controls use the target-action mechanism to respond to touch events. Since the iPhone is a Multi-Touch device, many different events can occur, such as tapping, multitapping, dragging, and releasing. Luckily, each control has been designed to respond to only those events appropriate for its usage, and each does so in an intuitive and consistent manner.

You'll now take a closer look at the controls available to iPhone developers.

Buttons

When adding buttons to your application, you'll use the **UIButton** class **B**. The default initializer for buttons is the **buttonWithType:** method:

UIButton *myButton =

- → [[UIButton buttonWithType:
- → UIButtonTypeRoundedRect];



B The default button types for **UIButton**.

To be notified when a button changes state, add a target and action:

[myButton addTarget:self action: → @selector(buttonClick:)

- → forControlEvents:UIControlEvent
- → TouchUpInside];

The **UIControlEventTouchUpInside** event is most commonly used for handling regular button presses.

The **UIButtonTypeCustom** type lets you create buttons with images or even draw them yourself using your own custom drawing code (as discussed earlier in the "Views" section).

To create a button with an image:

 Specify an image for the button's default state using NSControlStateNormal:

UIImage *buttonImage =

- \rightarrow [UIImage imageNamed:
- → @"myButtonImage.png"];

[myButton setImage:buttonImage → forState:UIControlStateNormal];

UIButton will automatically apply highlight effects to indicate that the button is pressed or disabled.

2. You can also set multiple appearance properties for each of these states, including the title text, font, and color.

You can use different images for the four different states: the default (as shown in step 1), highlighted, selected, and disabled. This enables you to create buttons to represent other controls.

To create a checkbox button:

1. Assign images for both of the buttons' states:

```
[checkbox setImage:[UIImage

→ imageNamed:@"checkbox_off.png"]

→ forState:UIControlStateNormal];
```

```
[checkbox setImage:[UIImage

→ imageNamed:@"checkbox_on.png"]

→ forState:UIControlStateSelected
];
```

2. Set the target method to call when the button is tapped:

```
[checkbox addTarget:self action:
```

- ightarrow @selector(checkboxClick:)
- \rightarrow forControlEvents:UIControlEvent
- ightarrow TouchUpInside];
- In the checkboxClick: method, simply flip the button's selected property:

```
btn.selected = !btn.selected;
```

Since you've previously defined images for the two different states, the button automatically updates to display the correct image. **Code Listing 4.29** shows the updated code.

IP If you specify an image or title for any button type other than UIButtonTypeRounded Rect, the button effectively becomes a button of UIButtonTypeCustom.

Switches

Switches, represented by the **UISwitch** class, let you create an on/off control **G**.

To create a switch:

1. Use the initWithFrame: method:

CGRect switchRect = CGRectMake → (120,50,0,0);

```
UISwitch *mySwitch = [[UISwitch
→ alloc] initWithFrame:switchRect];
```

Code Listing 4.29 Creating a check box.

```
- (void)checkboxClick:(UIButton *)sender {
    sender.selected = ! sender.selected;
3
- (void)viewDidLoad {
    UIButton *checkbox = [UIButton buttonWithType:
                          UIButtonTypeCustom];
    CGRect checkboxRect = CGRectMake(135,150,36,36);
    [checkbox setFrame:checkboxRect]:
    [checkbox setImage:[UIImage
                        imageNamed:@"checkbox_off.png"]
              forState:UIControlStateNormal];
    [checkbox setImage:[UIImage
                        imageNamed:@"checkbox_on.png"]
             forState:UIControlStateSelected];
    [checkbox addTarget:self
                 action:@selector(checkboxClick:)
       forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:checkbox];
}
```

Settings Sounds	
۰ 	
Ringtone	Old Phone >
New Text Message	Horn ≯
New Voicemail	ON
New Mail	OFF
Sent Mail	ON
Calendar Alerts	ON
Lock Sounds	ON
Keyboard Clicks	ON

C Switches are used extensively in the Settings application of the iPhone.



D The control hierarchy that makes up a **UISwitch**.

Since switches are always the same size, the width and height properties are ignored.

 When you change a switch's value, it generates a UIControlEventValueChanged event:

[mySwitch addTarget:self action: → @selector(switchAction:)

- → forControlEvents:UIControlEvent
- → ValueChanged];
- To turn a switch on/off, call the setOn:Animated: method:

[mySwitch setOn:YES animated:YES];

Switches don't have any properties for modifying the default visual appearance, but with a little digging, you can control a couple of elements.

Within the control hierarcy of a **UISwitch**, the "on" and "off" elements are **UILabels D**. You can manipulate the text, font, color, and more.

To alter the appearance of a switch:

 To retrieve the two UILabels within the switch that hold the switch's text, you can use this:

UIView *mainView = [[[[mySwitch → subviews] objectAtIndex:0] → subviews] objectAtIndex:2];

UILabel *onLabel = [[mainView → subviews] objectAtIndex:0];

UILabel *offLabel = [[mainView → subviews] objectAtIndex:1];

continues on next page

 Now you can change the text and color of these labels. The choice of text values is quite limited since the labels are small in size and are clipped by their containing view:

```
onLabel.text = @"YES";
offLabel.text = @"NO";
onLabel.textColor =
→ [UIColor yellow Color];
offLabel.textColor =
→ [UIColor green Color];
```

3. When setting the text values, you should localize your replacement text wherever possible. **Code Listing 4.30** shows the updated code.

Code Listing 4.30 Customizing the switch control.

```
-(void)switchAction:(id)sender
{
    NSLog(@"switch changed");
3
- (void)viewDidLoad {
    [super viewDidLoad];
    CGRect switchRect = CGRectMake(120,50,0,0);
    UISwitch *mySwitch = [[UISwitch alloc] initWithFrame:switchRect];
    [mySwitch addTarget:self action:@selector(switchAction:) forControlEvents:UIControlEventValueChanged];
    //customize the appearance
    UIView *mainView = [[[[mySwitch subviews] objectAtIndex:0] subviews] objectAtIndex:2];
    UILabel *onLabel = [[mainView subviews] objectAtIndex:0];
    UILabel *offLabel = [[mainView subviews] objectAtIndex:1];
    //change the text
    onLabel.text
                      = @"YES";
    offLabel.text
                      = @"NO";
    //change the text color
    onLabel.textColor = [UIColor yellowColor];
    offLabel.textColor = [UIColor greenColor];
    [self.view addSubview:mySwitch];
    [mySwitch release];
}
```



(E) The brightness slider control indicates the change in value with graphics at both ends of the control.



A custom slider control.

Sliders

Although switches have only two possible states, sliders let you select from a range of values on a horizontal bar, or *track*, with a *thumb* indicator that can be moved from side to side to select values **1**.

Unlike the **UISwitch**, there's quite a lot you can do to customize the visual appearance of sliders, such as putting images to represent the values at either end of the track. You can also customize the thumb image and the graphics that appear on the track on both sides of the thumb as the values change **(**).

Just as with the **UISwitch**, sliders create a **UIControlEventValueChanged** event when their value is changed. By setting the **continuous** property, you can choose to have these events fired either as the slider is changed or at the end of a change. **Code Listing 4.31** demonstrates this with a custom **UISlider**, with minimum, maximum, and thumb images. In the **sliderAction:** method, you are forcing a "step" behavior, making the slider jump to the next value in increments of ten. A label added to the view displays the current slider value.

(IP) Although not specified in the developer documentation, setting the thumb image of a UISlider also hides the tracking image. You must also set the minimum and maximum track images.

The stretchableImageWithLeft CapWidth:topCapHeight: method lets you create an image that can stretch in the center but does not stretch on either side, as shown in the rounded edges of the track images.

Code Listing 4.31 Implementing a custom slider.

```
UILabel *IblSliderValue:
@implementation UITestViewController
-(void)sliderAction:(id)sender
£
    int stepAmount = 10;
    float stepValue = (abs([(UISlider *)sender value]) / stepAmount) * stepAmount;
    [sender setValue:stepValue];
    lblSliderValue.text = [NSString stringWithFormat:@"%d",(int)stepValue];
3
- (void)viewDidLoad {
    [super viewDidLoad];
    CGRect sliderRect = CGRectMake(20,50,280,40);
   UISlider *mySlider = [[UISlider alloc] initWithFrame:sliderRect];
    mySlider.minimumValue = 0;
    mySlider.maximumValue = 100;
    mySlider.continuous = YES;
    //images
    UIImage *leftTrackImage = [[UIImage imageNamed:@"left_slider.png"] stretchableImageWithLeftCapWidth:5.0 topCapHeight:0.0];
    UIImage *rightTrackImage = [[UIImage imageNamed:@"right_slider.png"] stretchableImageWithLeftCapWidth:5.0 topCapHeight:0.0];
    UIImaae *thumbImaae
                           = [UIImage imageNamed:@"apple_thumb.png"];
    UIImage *minSliderImage = [UIImage imageNamed:@"apple_min.png"];
    UIImage *maxSliderImage = [UIImage imageNamed:@"apple_max.png"];
    [mySlider setThumbImage:thumbImage forState:UIControlStateNormal];
    [mySlider setMinimumTrackImage:leftTrackImage forState:UIControlStateNormal];
    [mySlider setMaximumTrackImage:rightTrackImage forState:UIControlStateNormal];
    [mySlider setMinimumValueImage:minSliderImage];
    [mySlider setMaximumValueImage:maxSliderImage];
    [mySlider setValue:50.0f];
    //handle value change events
    [mySlider addTarget:self action:@selector(sliderAction:) forControlEvents:UIControlEventValueChanged];
    //label to show current value
    CGRect lblRect = CGRectMake(145,100,100,20);
    lblSliderValue = [[UILabel alloc] initWithFrame:lblRect];
    lblSliderValue.backgroundColor = [UIColor clearColor];
    lblSliderValue.text = [NSString stringWithFormat:@"%d",(int)mySlider.value];
    //add slider to main view
    [self.view addSubview:mySlider];
    [self.view addSubview:lblSliderValue];
    [lblSliderValue release];
    [mySlider release];
3
```



Segmented controls

The **UISegmentedControl** consists of a horizontal control divided into segments **③**. Segmented controls are useful for allowing users to pick from a group or set of values.

Each segment functions as its own button. By default, selecting a segment will deselect the others in the control (much as a radio button does in HTML). You can alter this behavior by setting the **momentary** property.

To create a segmented control:

- Create an array of UIImages or NSStrings, and then call the default initializer initWithItems: (Code Listing 4.32).
- **2.** Set the frame, and the control will automatically resize to accommodate its segments.

Each segment will initially be the same size.

continues on next page

Code Listing 4.32 Creating a segment control.

 Set the width of individual segments using the setWidth:forSegmentIndex: method.

This will automatically resize any other segments that have not had their widths explicitly set to fit within the control.

- Select segments using the setSelected SegmentIndex: method.
- Disable individual segments using the setEnabled:forSegementAtIndex: method.
- 6. Add more segments using insertSegments WithImage:atIndex: animated:

or

insertSegmentsWithTitle:atIndex:
animated:.

- 7. Set the **animated** property to **YES** so your segments will "slide in" as they are added.
- 8. To remove segments, use the remove SegmentsAtIndex:animated: method.
- 9. Use **removeAllSegments** to clear the entire control.

Segment control styles

Segment controls have three different styles, which can be set using the **segmentedControlStyle** property **①**. Set the style to **UISegmentedControlStyleBar** to change the color of the control via the **tintColor** property, but depending on the color you use, you may not be able to see the difference between selected and unselected. **Code Listing 4.33** shows an example of how to use some of the properties of a segmented control.

Use the **UISegmentControl** to create a "glass" alternative to a **UIButton**. Use the **tintColor** property to change the color of the button.



He three styles available for segmented controls.

Code Listing 4.33 Setting some of the segment control properties.

```
- (void)segmentClick:(id)sender {
    NSLog(@"clicked: %d",[sender selectedSegmentIndex]);
}
- (void)viewDidLoad {
    NSArray *arrSegments = [[NSArray alloc] initWithObjects:
                             [NSString stringWithString:@"0"],
                            [NSString stringWithString:@"1"],
                            [NSString stringWithString:@"2"],
                            nil];
    UISegmentedControl *mySegment = [[UISegmentedControl alloc]
                                      initWithItems:arrSegments];
    CGRect segmentRect = CGRectMake(10,50,300,40);
    [mySegment setFrame:segmentRect];
    [mySegment addTarget:self
                  action:@selector(segmentClick:)
        forControlEvents:UIControlEventValueChanged];
    [mySegment setSegmentedControlStyle:UISegmentedControlStyleBar];
    [mySegment setTintColor:[UIColor darkGrayColor]];
    //select first item
    [mySegment setSelectedSegmentIndex:0];
    //change a segment size
    [mySegment setWidth:120.0 forSegmentAtIndex:1];
    //add a new segment
    [mySegment insertSegmentWithTitle:@"new" atIndex:2 animated:YES];
    [self.view addSubview:mySegment];
    [arrSegments release];
    [mySegment release];
}
```

This page intentionally left blank

Index

Symbols

: (colon), using with methods, 7 { } (braces), using in header (.h) file, 3 + (plus) sign, prefixing class methods with, 7 [] (square brackets), using with methods, 5

A

ABGroup type, using with Address Book, 406 ABPerson type, using with Address Book, 406 accelerometer creating tilt-sensitive applications, 307-310 detecting shakes, 298-299 determining orientation, 299, 301-302 redrawing for orientation changes, 303 responding to, 307 updating for autorotation, 304-306 accelerometer data, smoothing out, 307 accessibility enabling via VoiceOver, 98–99 overview of, 98 accessibility attributes Frame, 101 Hint, 101 Label, 101 Traits, 101 Value, 101 Accessibility inspector, using with IB (Interface Builder), 102 accessible applications, creating, 99-100 action sheets versus alert views, 145 changing appearance of, 145 creating, 144-145

actions confirming, 144-145 creating in IB (Interface Builder) ma manually, 72-75 activity indicator view, creating, 140-141 Address Book creating contacts, 413-417 creating multivalue properties, 414-415 getting reference to, 407 group records, 410 group records stored in, 406 kABPerson* properties, 411 logging records to console, 407, 409 person records, 411–412 person records stored in. 406 record ID of records, 408 retrieving contact records in, 406-408 retrieving multivalue properties, 412 retrieving records from, 406-409 setting values for addresses, 415 setting values for phone numbers, 414 Address Book UI framework adding contacts, 424 adding labels for contact names, 423 adding to projects, 419 displaying contact information, 424 editing people, 421-427 people picker, 418-420 tapping contact images, 424 AddressBookExampleViewController.h file, 406 addresses, displaying in map annotations, 339 alarms, setting in calendar, 432 alert messages, displaying, 142–143 alert views versus action sheets, 145

alloc method calling, 13 using to create objects, 10 annotation callouts, altering, 337 annotation class, creating for maps, 334 annotations, displaying addresses in, 339 <app>Info.plist file contents of 44-45 settings. 87–88 Apple Developer Connection Web site, 41, 307 application delegate, using, 84-86 application icons key for, 88 removing "gloss effect" from, 88 application name, displaying, 88 application settings adding controls, 92–93 application preferences, 90 creating settings pages, 91 user preferences, 87–90 application setup, setting unique identifier for, 88 applications. See also interapp communication; projects; settings page adding classes to, 57 adding strings file to, 95-96 adding views to, 113 building in Xcode IDE, 58 creating settings page for, 91 with custom URL scheme, 104-105 force-quitting, 90 launching from other applications, 103 localizing, 95-97 making accessible, 99–100 peer-to-peer, 271-273 peer-to-peer chat, 273–278 running in Xcode IDE, 58 setting preferences for, 90 setting version numbers for, 88 sharing information between, 105, 109-110 shipping with default settings files, 239-243 terminating in background, 88 updating in IB (Interface Builder), 70–72 view-based, 312

arravs accessing objects in, 25 creating, 24 creating for parsed data, 256 getting lengths of, 24 looping back through values of, 25 mutable, 26 sorting for strings, 25 using @selector keyword with, 25 using with drill-down details application, 220 using with table views, 204-205 verifying objects in, 25 articles array, using with parsed RSS feed, 255 asynchronous connections updating applications for, 251–253 using, 251 audio. See also background audio controlling from background, 361–365 creating Play button for, 368 playing, 398 playing from iPod library, 399 playing in background, 358-360 recording, 366-370 audio controls, accessing, 362 audio events, responding to, 356 audio player application adding controls to, 352-356 checking playing status of, 352 completing, 354-355 creating, 350-351 creating user interface, 352 implementing delegate methods, 357 implementing play: method, 352 implementing scrub: method, 354 implementing stop method, 352 preloading buffers, 352 resetting audio controls, 357 responding to events, 356-357 setting volume for, 352-353 setting volume property, 354 audio recorders passing nil to settings parameter, 368, 370 settings for, 368

audio session, setting up, 359 audio settings, configuring, 368 AudioPlaverExampleViewController.h file, 350. 352, 356, 358 AudioRecorderExample.h file, 366 autorelease pools, using in memory management, 11–13 autoresizingMask values, described, 118 autorotation defined. 303 using, 304-306 AVAudioPlayer class behavior of, 351 explained, 350 AVAudioPlayerDelegate protocol adopting, 356 implementing, 396 implementing methods for, 357

B

background audio, controlling volume of, 362. See also audio background audio service, 461 background color changing for orientation, 302 setting for movies, 389 background location service, 459-461. See also location manager background mode, entering and exiting, 457-458 battery power, preserving for location manager, 315.319 birthdays, creating events for, 431-433 bookmarks, using in Xcode IDE, 53 border styles, using with text fields, 156 bounds, using with views, 113-114 braces ({ }), using in header (.h) file, 3 breakpoints adding in Xcode IDE, 52 removing in Xcode IDE, 52 brightness slider, 175 Build Results window, displaying in Xcode IDE, 58

buttons adding for alert views, 142–143 adding to applications, 170–171 adding to main view, 184 adding to toolbars, 153–156 checkbox, 172 creating for custom cells, 227 creating for taking photos and videos, 377 predefined, 153 specifying images for, 171 using target-action pattern with, 37

С

calendar event store, querving, 429 calendar events editina, 438-442 viewina, 434-437 calendars, using with dates, 22. See also iPhone calendar calendars property, passing nil in, 429 camel case notation, variation of, 5 camera application, 378-379 camera mode, launching image picker in, 378 camera support, checking for, 376 capitalization, setting for keyboards, 158 categories, using as alternative to subclassing, 37 - 38cells, customizing in table views, 224-232 CGRect converting to NSString, 114 creating for frame of view, 113 creating for subview, 117 uses of. 286 checkbox buttons, creating, 172 class files, adding in Xcode IDE, 57 class methods autoreleased objects returned by. 6 defining, 6 prefixing with plus (+) sign, 6 usina. 6 classes. See also subclassing adding to applications, 57 arrays, 24-26 class methods provided by, 6

classes (continued) dates and times, 20–23 dictionaries, 27-29 header (.h) file, 3 implementation (.m) file, 4 methods, 5-7 with methods for handling files, 234–235 notifications, 30-31 saving in Xcode IDE, 57 strings, 14-20 timers, 32-34 clicks, capturing in web views, 169 **CLLocation** events generating, 312 properties of, 318 **CLLocationManager** class, delegate methods, 314 CLPreviewController object, using, 244 Cocoa, defined, 1 Cocoa Touch Class, adding, 57 Cocoa Touch framework group, described, 2 code collapsed in Xcode IDE, 52 commenting in Xcode IDE, 53 displaying in functional groups, 52 hiding in Xcode IDE, 52 uncommenting in Xcode IDE, 53 code completion, using in Xcode IDE, 53 code-signing identities, resource for, 79 colon (:), using with methods, 7 Command key. See keyboard shortcuts compass, accessing in Core Location framework, 323-324 contact information displaying, 424 editing in Address Book, 425 contact names, adding labels for, 423 contact records, retrieving in Address Book, 406-408 ContactExampleViewController.h file, 413 contacts adding to Address Book, 424 creating for Address Book, 413-417 grouping in Address Book, 410

Contacts application displaying people picker in, 418 viewing contacts in, 416 Control key. See keyboard shortcuts controls buttons, 170-171 defined, 111 segmented, 177-179 sliders, 175–176 switches, 172-173 using UIControl class with, 170 copy method, calling, 13 Core Location framework. See also location manager accessing compass, 323-324 adding logging of data to screen, 316-317 adding timeouts, 318-323 adding to projects, 327 CLLocation events, 312 CLLocationManager class, 312 decreasing desiredAccuracy level, 323 handling location updates, 314-315 increasing accuracy, 317-318 power used by, 315 testing outside simulator, 315-316 Core OS framework group, described, 2 Core Services framework group, described, 2 CoreLocationExampleViewController.h file, 312, 316, 318

D

data detectors, using with text views, 161 date and time intervals, calculating, 20–21 date objects, creating for calendar, 428 date picker, creating, 151 dates calculating seconds between, 20–21 comparing, 20–21 creating, 20 getting day, month, and year from, 21 localizing, 94 setting styles for, 23

Debugger Console displaying location information in. 314 logging tap counts to, 286 logging touch locations to, 294 defaults system, reading and writing to, 88 Delegate design pattern, using, 36 delegate methods for CLLocationManager class, 314 defining for picker view, 148 implementing for alert view, 143, 162-163 implementing for page control, 133 implementing for zooming, 131 for multitasking, 458 using to manage heading updates, 324 for web pages, 165 design patterns categories, 37-38 Delegate, 36 MVC (Model View Controller), 35 singletons, 39 Target-Action, 37 DetailViewController.h file, 218 developer, registering with Apple as, 41 dictionaries accessing objects in, 28 creating, 27 mutable, 29 populating from files, 28 verifying number of elements in, 28 directories Documents, 237 Library, 238 tmp, 237 documentation viewer, launching for iOS, 78 documents, previewing, 244 Documents directory, explained, 237 double-tap support, adding, 287 drill-down details application, main header file for, 219

Ε

editor pane in Xcode IDE, using, 50–51 **EKEventViewController**, using, 436–437 e-mail attaching files to, 447 composing and sending, 443–449 showing compose interface for, 447 e-mail libraries, using, 444 EmailExampleViewController.h file, 444 event store adding events to, 432 creating for birthdays, 431–433

F

file sharing, enabling, 238 file system, overview of, 236 files adding to projects, 51 attaching to e-mail. 447 classes related to, 234 copying, 236 creating in Xcode IDE, 57 editing in Xcode Organizer, 79 finding quickly in projects, 51 opening from keyboard, 51 reading in application bundles, 238 FilesExampleViewController.h file, 240 "flip" animation, using with modal views, 189-192 FlipModalExampleViewController.h file, 189 focus ribbon in Xcode IDE, explained, 52 font size, specifying for labels, 136 fonts, list of, 137 force-quitting applications, 90 Frame accessibility attribute, 101 frames, representing views as, 112 frameworks adding to projects, 2 defined, 2 referencing in code, 2 French localization, displaying, 97 French strings file, adding to application, 95-96

Game Kit API, using for peer-to-peer applications, 271–273 Get Info in Xcode IDE, keyboard shortcut, 46–47 getter methods, generating, 9 GetWebContentViewController.h file, 248, 251 graphics adding for load screen, 79 adding to tabs, 196 using in image views, 126 groups in Xcode creating, 45 static, 44 gutter in Xcode IDE, explained, 52

H

header (.h) file @end directive, 3 **#import** directive, 3 @interface line, 3 use of braces ({ }) in, 3 heading updates, checking support for, 324 helloWorld target in Xcode, features of, 46-47 helloWorldViewController.h file. 72 helloWorldViewController.xib file construction of 76-77 opening in IB, 65 Hint accessibility attribute, setting, 100-101 home directory, contents of, 237 HTTP authentication, responding to, 266 hyperlinks handling in web views, 168-169 opening in Safari application, 169

I

IB (Interface Builder) Accessibility inspector, 102 actions, 64 configuring slider in, 74 creating actions manually, 72–75 creating interfaces, 69–70 creating outlets manually, 72–75 displaying class's actions, 70

displaying class's outlets, 70 document window, 65-66 features of, 64-65 File's Owner object, 65 First Responder object, 66 increasing width of time label, 73 inspector window, 67-69 laying out applications, 69 Library window, 67 outlets, 64 updating applications, 70–72 using to set styles for table views, 207 View object, 66 XIB files, 65 image picker closing, 374 creating application with, 372-375 hiding, 374 launching in camera mode, 378 setting **sourceType** property of, 371–372 image views. See also views controlling scrolling behavior of, 127 creating, 126 resizing, 126 using with scroll views, 129-130 ImagePickerExample.h file, 372 images animating, 127 animating over, 128 choosing from photo library, 371 displaying as annotations, 336-337 getting paths of, 238 panning around, 130 specifying for buttons, 171 using scroll view with, 129-130 zooming in and out of, 130 implementation (.m) file @end directive, 4 @implementation line, 4 **#import** directive, 4 @synthesize directive, 4 interapp communication, using openURL: method in, 103. See also applications interface, redrawing when rotating, 303

Interface Builder (IB) Accessibility inspector, 102 actions, 64 configuring slider in, 74 creating actions manually, 72-75 creating interfaces, 69-70 creating outlets manually, 72-75 displaying class's actions, 70 displaving class's outlets, 70 document window, 65-66 features of, 64–65 File's Owner object, 65 First Responder object, 66 increasing width of time label, 73 inspector window, 67-69 laving out applications, 69 Library window, 67 outlets, 64 updating applications, 70-72 using to set styles for table views, 207 View object, 66 XIB files, 65 Internet connections, testing for, 252 iOS SDK (software development kit) documentation, 78 downloading, 41 frameworks, 2 iPhone, displaying logging information on, 317 iPhone calendar. See also calendars accessing database for, 428 adding events, 430 alarms event, 430 calendar event, 430 creating date objects, 428 creating events for birthdays, 431–433 editing events, 438-442 Event Kit UI classes, 434 eventIdentifier event, 430 location event, 430 notes event. 430 recurrenceRule event, 430 retrieving events from, 428-429 setting alarms, 432 startDate/endDate event, 430

title event. 430 viewing calendar events, 434-437 viewing details of events, 436-437 viewing event details, 434 Phone screen, adding logging location data to. 316 - 317iPhone Simulator adding photos to, 63 backing up data on, 63 features of, 61-63 limitations of 61–62 removing applications from, 63 resetting, 63 iPhone vs. iPhone Simulator, building for, 58 iPhones, identifying via UDID, 79 iPod library accessing media items, 392 playing audio from, 399 selecting songs from, 397 iPodLibraryExample.h file, 396, 399-400 iPods, identifying via UDID, 79 iTunes file sharing, support for, 88

J

JavaScript, executing in web views, 167

Κ

keyboard shortcuts bookmarks in Xcode IDE, 53 building applications in Xcode IDE, 58 code management in Xcode IDE, 52 documentation viewer. 78 finding text in Xcode IDE. 53 force-quitting applications, 90 Get Info in Xcode IDE. 46-47 help in Xcode IDE, 53 jump-to-definition in Xcode IDE, 53 opening files in windows (Xcode), 51 Project Find window in Xcode IDE, 53 Redo in Xcode IDE. 60 Single File Find in Xcode IDE, 53 Undo in Xcode IDE. 60 Xcode IDE. 60

keyboards hiding, 161 scrolling interface in response to, 162–163 setting capitalization for, 158 using, 157–159 keyboardType property

disabling, 158 setting, 157

L

Label accessibility attribute, 101 label text adding shadows to, 136 alianina, 138 displaving lines of, 138 labels controlling wrapping of, 138 creating and setting properties of, 136, 138 setting line counts for, 138 specifying font sizes for, 136 landscape versus portrait orientation, 186, 303. 305-306. See also orientation language codes, resource for, 97 launch image, specifying name of, 88 launchOptions values, displaying, 107 layoutSubviews method, using with table views. 225 Library directory, explained, 238 links handling in web views, 168-169 opening in Safari application, 169 load screen, adding graphic for, 79 load time, speeding perception of, 79 loadView method, explained, 183 local notifications. See also notifications creating application for, 462–465 responding to, 466-468 service, 461 localization dates, 94 numbers. 94 overview of, 94-95 support for, 88

localized applications, creating, 95-97 location aware applications, creating, 312–313 location events checking ages of, 320 filtering, 317-318 location manager. See also background location service; Core Location framework adding timeouts to, 318-323 generating, 319 heading Available class method, 324 location search, results of, 323 location updates, handling, 314–315 locations setting and showing on maps, 328 showing in maps, 328 logging information, displaying on iPhone, 317 long-touch support, adding, 288-291 low-memory conditions, handling, 193. See also memory management

Μ

.m (implementation) file @end directive. 4 @implementation line, 4 **#import** directive, 4 @synthesize directive, 4 Mail application, launching, 103, 443 MainWindow.xib, objects for, 76-77 map center coordinate, creating variable for, 328 Map Kit framework, adding to projects, 325 map overlays, creating, 329-330 mapping application adding helper methods, 344 adding instance variables to, 342 completing, 345-348 creating **CGRect** for address view. 343 header file for. 342 implementing delegates for, 344 MappingExampleViewController.h file, 325, 330, 338, 341 maps adding annotations to, 333-337 adding reverse geocoding to, 338–340

adding to applications, 325-327 defining regions on, 330 displaying, 329 displaying addresses in annotations, 339 drawing routes on, 330-332 drawing shapes on, 329-330 removing annotations from, 337 showing locations on, 327-328 zooming into, 328 Maps application, launching and searching, 104 mapType property, using to display maps, 329 media collections, accessing, 394-395 Media framework group, described, 2 media items accessing in iPod library, 392–393 metadata properties for, 394 playing, 398 media picker adding, 396-397 closing, 397 media player, creating, 400-404 media guery, console output of, 393-394 MediaPlayer framework, adding, 382 memory management. See also low-memory conditions autorelease pools, 11–13 referencing counting, 10 MessageUI framework, using, 443-444 messaging methods, 5 method calls nesting, 7 performing steps in, 8 methods calling, 5-6 initializer. 8 passing values into, 5 phrases used with, 5 syntax for, 6 use of square brackets ([]) with, 5 using colon (:) with, 7 writing, 7 MKCircle class, using with maps, 330 MKMapView, example of, 326 MKPolygon class, using with maps, 330

modal views, displaying, 189–193 Model View Controller (MVC) design pattern, 35 motion* methods, using to detect shakes, 298 movement detection. See accelerometer movie playback, controlling, 386 movie player, customizing, 387-390 movie player video controller completed code, 385 header file for. 382 using, 384 movie recording, time limitation of, 375 MoviePlayerExampleViewController.h file, 382 387 movies. See also videos loading from network locations, 390-391 setting background color of, 389 showing activity indicator for, 390 MPMediaItem class, explained, 392 MPMediaPickerController view controller class, 396 MPMoviePlayerController:, using, 387-391 MPMusicPlayerController class, 398 multitasking delegate methods, 458 overview of, 456 preventing, 457 verifying capability for, 457 multitasking services background audio, 461 background location, 459-461 local notifications, 461 task completion, 459 VoIP (Voice over IP), 461 multi-touch gestures. See also touch-based applications pinch, 292-294 rotate, 292-294 supporting, 292-294 zoom, 292-294 MultiTouchExampleViewController.h file, 292,295 music players, types of, 398 MVC (Model View Controller) design pattern, 35 MyCustomCell.h file, 224

Ν

Name property, using with settings page, 93 navigation bar in Xcode IDE, using, 55–56 navigation controllers, using with table views, 217 network locations, loading movies from, 390-391 networking creating peer-to-peer applications, 271–273 creating peer-to-peer chat applications, 273-278 parsing RSS feeds, 255-261 parsing XML, 254 responding to HTTP authentication, 266 retrieving content from web pages, 248 retrieving stock quotes from web pages, 248-251 searching Wikipedia, 262–265 sending data to web pages, 262 updating status on Twitter, 266–271 using asynchronous connections, 251–253 NIB (NeXT Interface Builder), 65 NIB Name property, using with MainWindow.xib. 76 notifications. See also local notifications described, 30 registering objects as observers for, 30 using, 30-31 NSCalendar class, using, 21 NSData class, using with files, 234 NSDate class, using, 20 NSDate objects, creating, 21 NSDateFormatter class, using, 22-23, 94 NSDictionarv loading and saving as file, 239-243 usina. 27 using with files, 234 NSFileManager class, explained, 236 NSHomeDirectory() function, using, 237 NSLog() statements using with asynchronous connections, 252-253 using with stock quotes, 250 NSMutableArray class, using, 26

NSMutableDictionary class, using, 29 NSNotification object, using, 30 NSNumberFormatter class, using, 94 NSString class containing numbers, 15 converting CGRect to, 114 creating, 14 file functions, 18 format specifiers, 14 immutable quality of, 14 initializer methods in, 8 stringWithContentsOfURL: method, 248 using to read and write to URL, 18 using with files, 234 NSTimer class, using, 32 NSURLConnection class, using, 251–252 NSUserDefaults class, using, 87, 90 NSXMLParser class, using, 254, 256 numbers, localizing, 94

0

Objective-C classes. 3-4 creating objects, 7-8 defined.1 methods. 5-7 properties, 8–9 obiects calling release method for, 10-11 creating, 7-8 on/off controls, creating, 172-173 openURL: method, using with UIApplication class, 103 Option key. See keyboard shortcuts orientation. See also landscape mode versus portrait orientation; view controllers autorotating, 303 detecting, 299, 301-302 determining for shakes, 299-301 Portrait versus Landscape, 186 responding to changes in, 184-188 specifying for applications, 88 tracking changes in, 187 orientation changes, redrawing interface for, 303 OrientationExampleViewController.h file, 304 outlets, creating in IB (Interface Builder) manually, 72–75 Overview toolbar, using in Xcode IDE, 59

P

P2PExampleViewController.h file, 274 page control, creating for scroll view, 132–135 parser delegate methods, implementing, 256 parser variable, using with RSS feed, 255 passwords, saving in settings file, 239-242 pasteboard, using, 109–110 paths getting array of filenames for, 236 retrieving for applications, 237 patterns. See design patterns PDF viewer, creating, 244-247 peer picker controller, 272 peer picker delegate method, implementing, 275 peer-to-peer applications, creating, 271–273 peer-to-peer chat application, creating, 273-278 people picker adding to Address Book, 418–420 editing in Address Book, 421–427 updates for editing contacts, 426–427 PeoplePickerExampleViewController.h. 419, 421 phone numbers dialing, 104 setting values in Address Book, 414 photo library choosing images from, 371 determining empty status of, 375 photos adding to iPhone Simulator, 63 taking, 375–380 picker views. See also views creating, 146–148 enhancing, 148–150 pictures. See images: photos pinch gestures, adding, 295–297 Play button, creating for audio, 368 playback queue, using with audio, 398

playlists, retrieving, 395 plus (+) sign, prefixing class methods with, 6 portrait versus landscape orientation, 186, 303. 305–306. See also orientation PostTweetViewController.h file. 266 PostWebContentController. h file, 262 predicate, creating for search text, 214 preferences application, 90 user, 87, 89-90 PreferenceSpecifiers key, using, 91 progress views, creating, 139–140 Project Find history, accessing in Xcode IDE, 53 projects. See also applications adding files to, 51 adding frameworks to, 2 creating in Xcode IDE, 43 properties defining, 9 using getter methods with, 8 using setter methods with, 8 property list files, selecting in Xcode IDE, 50 provisioning profiles, use of, 79, 82

Q

QuartzCore framework, adding to long-touch project, 289 Quick Look framework adding, 244 resource for, 246

R

Record button, creating for audio, 368 recording settings, controlling, 370 rectangles. See CGRect referencing counting, using in memory management, 10 release method, calling for objects, 10–11 responder objects, defined, 280 retain method, calling, 13 Return key changing text on, 158 hiding text field for, 158 reverse geocoding, adding to maps, 338–340 RootViewController.h file, 201, 204, 210, 218, 225, 434 rotate gestures, adding, 295–297 rotating iPhones, 303 views, 296–297 rotation transforms, applying to views, 124–125 **roundedCornerView** class, creating, 120–122 routes, drawing on maps, 330–332 rows grouping into sections and styles, 204 indicating for custom cells, 230 RSS feeds format of XML records for, 254 parsing, 255–261

S

Safari application, opening links in, 169 sandbox, defined, 233 saveClick: method, implementing, 241–242 scale transforms, using with views, 123–125 scope highlighting effect, using in Xcode IDE, 52 screen, adding logging location data to, 316-317 scroll views adding zoom to, 131 paging content of, 131 using to zoom in and out of images, 130 using with images, 129–130 search text, creating predicate for, 214 seamented controls creating, 177-178 properties, 179 removing, 178 styles, 179 setter methods, generating, 9 Settings example, header for, 240 settings file saving password in, 239-242 saving username in. 239–242 settings page adding controls to, 92-93 creating for applications, 91 setting Name property for, 93

setting Title property for, 93 setView: method, using with table views, 224 ShakeExampleViewController.m file, 299 shakes detecting, 298 determining orientation for, 299-301 supporting, 299 Shift key. See keyboard shortcuts simulator. See iPhone Simulator single-tap support, adding, 287 singletons, using, 39 skpsmtpmessage class, availability of, 444 slider, configuring in IB (Interface Builder), 74 sliders features of. 175 implementing, 175–176 smart groups, creating in Xcode IDE, 48 SMS (Short Message Service) composing and sending, 450-454 creating body of, 452 SMS application, launching, 103, 450 SMSExampleViewController.h file, 451 songs, selecting from iPod library, 397 splash screens, adding to applications, 86 square brackets ([]), using with methods, 5 startWiggle: method, using with long-touch support, 289 status bar choosing display styles for, 88 launching applications without, 88 leaving visible, 88 stock quotes, retrieving from web pages, 248-251 **stopWiggle:** method, using with long-touch support, 289 string methods, common uses of, 19 strings combining, 17 comparing, 15 converting case of, 15 creating arrays with substrings, 17 creating substrings, 16 getting lengths of, 15 NSString class, 14–19

performing case-sensitive comparisons, 15 replacing substrings in, 17 searching for substrings in, 17 trimming characters from, 16 verifvina substrinas in. 17 strings file, adding to applications, 95–96 stringWithContentsOfURL: method, using, 248 subclassing, alternative to, 37-38. See also classes substrings. See strings switches altering appearance of, 173–174 creating, 172-173 creating for custom cells, 227 synchronous connection, explained, 250 system items availability of, 153 types of, 154

T

tab bar items hiding, 217 updating applications for use of, 196–199 tab view controllers, implementation files for, 198-199 tab views, using 194–199. See also views table views. See also views cells in. 200 creating applications with, 201-203 creating arrays for sections of, 204–205 creating predicate for search text, 214 creating with drill-down behavior, 218–222 customizina cells in. 224–232 drilling down in, 217 editable for searching, 210-216 editing, 210 elements of, 200 grouped, 204-209 grouping rows in, 204 implementing data sources for, 200 implementing delegates for, 200 searching, 210 setting styles in IB (Interface Builder), 207 styles for cells, 223

suppressing delete button, 214 UITableView class, 200 UITableViewCell class, 200 using, 200 using navigation controllers with, 217 using sections with, 204-209 tabs adding graphics to, 196 adding titles to, 196 creating applications with, 194–195 disabling, 197 limiting number of, 195 selecting in code, 195 tap counts, logging to Debugger Console, 286 tap delay, setting, 288-289 tapCount property, using with touch objects, 285 tappable links, converting data into, 161 tapping support adding, 285-286 single and double, 287–288 target-action pattern defined, 111 using, 37 targets in Xcode IDE cleaning, 58 helloWorld, 46-47 using, 46-47 task completion service, 459 templates, choosing in Xcode IDE, 42–43 text fields border styles for, 156 creating, 156-158 hiding for Return key, 158 removing text from, 157 resizing automatically, 157 restricting content entered into, 159-160 text views using, 160 using data detectors with, 161 thumb image, setting for UISlider, 175 TiltingExampleViewController.h file, 307 tilt-sensitive applications, creating, 307–310

time. See date and time intervals: World Clock application time picker, creating, 151 timers creating, 32, 34 stopping, 33 Title property, using with settings page, 93 titles, adding to tabs, 196 tmp directory, explained, 237 toolbar items, creating, 154–155 toolbars adding buttons to, 153-156 autoresizingMask property of, 152 creating, 152-153 sizing, 154 updating in Xcode IDE, 49 touch locations, logging to Debugger Console, 294 touch-based applications. See also multi-touch gestures adding long-touch support, 288-291 creating, 281-283 header file of. 281 updated header file, 283 updating, 283-285 touch-based events, methods for, 280 TouchExampleViewController.h file, 281, 283, 287, 289 Traits accessibility attribute, 101 tweetClick: method, implementing, 267 Twitter, updating status on, 266-271

U

UDID (unique device identifier), 79 UIApplication class, using openURL: method with, 103 UIApplicationDelegate class, using, 84 UIBarButtonItem class, using, 153–156 UIButton class, using, 170–171 UIControl class, using, 170 UIDatePicker class, using, 151 UIDevice class, using, 86 UIDevice singleton, using with shakes, 299 UIImage class, using with files, 234

UIImagePickerController class, using, 371 UIImageView class, using, 126-127 **UILabel** class instances of, 136 retrieving for switches, 173-174 UIPageControl, using with scroll views, 131 UISegmentedControl class, described, 177 UISlider, setting thumb image of, 175 UISwitch class, using, 172-173 UITabBarController class, using, 194 UITableView class, explained, 200 UITextField class, using, 156 UITextView class usina, 160 using with stock quotes, 249 UIToolbar class, using, 152–153 UITouch object, explained, 280 **UIView** class creating subclasses for, 118 usina, 112 UIViewController class, described, 182 Unicode Consortium homepage Web site, 97 unique device identifier (UDID), 79 URL identifiers, support for, 88 **URL** scheme responding to being launched via, 106–108 using in interapp communication, 103–105 user preferences setting, 87, 89 storage of, 87 username, saving in settings file, 239-242

V

Value accessibility attribute, 101 version number, setting for applications, 88 vibrate feature, adding, 143 video button, availability of, 380 videos. *See also* movies playing, 381–385 taking, 375–380 view controllers. *See also* orientation displaying modal views, 189–192 main views, 182 responsibilities of, 182

UIViewController class, 182 view-based applications, creating, 312 viewDidAppear method, explained, 183 viewDidDisappear method, explained, 184 viewDidLoad method explained, 183–184 implementing, 241 viewDidUnload method, explained, 183 views. See also image views: picker views: tab views; table views; web views activity indicator, 140–141 adding subviews to, 113, 117-118 adding to applications, 113 alert, 142-143, 145 animating, 289 animating properties of, 115-117 applying rotation transform to, 124–125 applying scale transform to, 124–125 autosizing, 117-119 bounds, 113-114 creating custom rounded-corner, 120–122 custom drawing, 118 defined, 111 locating origins of, 112 nesting, 112 overview of, 112 presenting, 183-184 presenting modally, 191 replacing with custom classes, 122 representing as frames, 112 resizing for animations, 116 resizing via scale transform, 123 rotating, 296-297 rounded-corner, 120-123 specifying origins and sizes of, 112 text, 160 viewWillAppear method, explained, 183 viewWillDisappear method, explained, 184 VoiceOver enabling over iPhone, 98-99 improving descriptions used by, 100-101 VoIP (Voice over IP) service, 461 volume, controlling for background audio, 362

W

Web pages delegate methods for, 165 displaying in applications, 164-166 retrievina, 251 retrieving content from, 248 retrieving stock guotes from, 248-251 sending data to. 262 using Backward buttons in. 166 using Forward buttons in. 166 Web sites Apple Developer Connection, 41, 307 code-signing identities, 79 documentation for iOS. 78 iOS SDK 41 language codes, 97 registering as Apple developer, 41 skpsmtpmessage class, 444 Unicode Consortium homepage, 97 web views. See also views capturing clicks in, 169 executing JavaScript in, 167 handling hyperlinks, 168–169 implementing, 166 loading local content, 168–169 Wi-Fi, setting property for, 88 Wikipedia, searching, 262-265 World Clock application, 194

X

Xcode IDE Action toolbar feature, 48 adding classes to applications, 57 <app>-Info.plist properties, 44–45 bookmarks, 53 Bookmarks pop-up menu, 56 Bookmarks smart group, 45 Breakpoints pop-up menu, 56 Build and Go toolbar feature, 48 building and running applications, 58 building for iPhone vs. Iphone Simulator, 58 choosing project templates in, 42–43 Class hierarchy pop-up menu, 56 Xcode IDE (continued) cleaning targets, 59 code completion, 53 collapsed code in, 52 commenting code, 53 Counterpart pop-up menu, 56 creating files, 57 creating groups in, 45 creating projects in, 43 creating smart groups, 48 Debug configuration, 47 default smart groups in, 45 details pane, 49–50 displaying breakpoints in, 52 displaying errors in, 52 displaying line numbers in, 52 displaying warnings in, 52 editor pane, 50–51 Errors and Warnings smart group, 45 Executables smart group, 45 Find Results smart group, 45 find-and-replace operations, 53 Groups & Files pane, 44-45, 47 gutter and focus ribbon, 52 help feature, 53 hiding code in, 52 Included files pop-up menu, 56 Info toolbar feature, 48 jump-to-definition, 53 keyboard shortcuts, 60 Lock pop-up menu, 56 navigation bar, 55–56 opening files in windows, 51 overview of, 41-42 Overview toolbar, 59 Overview toolbar feature, 48 Project Find history, 53 Project Find window, 53 Redo keyboard shortcut, 60 Release configuration, 47 saving classes, 57 saving projects in, 43

SCM (source-code management) smart group, 45 scope highlighting effect, 52 Search toolbar feature, 48 selecting property list files, 50 Single File Find dialog box, 53 static groups, 44 targets, 46-47 Targets smart group, 45 Tasks toolbar feature, 48 toolbar features, 48 uncommenting code, 53 Undo keyboard shortcut, 60 updating toolbar in, 49 Xcode Organizer Archived Applications feature, 81 Developer Profile feature, 82 Device Logs feature, 82 Devices section, 80 editing files in, 80 iPhone development area, 81-82 Provisioning Profiles feature, 82 Screenshots feature, 82 Share Application feature, 81 Sharing section, 81 Software Images feature, 82 Submit to iTunesConnect feature, 82 Validate Application feature, 81 XIB files changing view mode of, 66 displaying two of, 76–77 using with IB (Interface Builder), 65 XML, parsing, 254 XMLExampleViewController.h file, 255

Z

zoom gestures, adding, 295–297 zooming adding to scroll view, 131 enabling, 131 into maps, 328 in and out of images, 130