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!

VISUAL QUICKSTART GUIDE

# 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.



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



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

*This page intentionally left blank*

# Table of Contents

# 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 high-definition (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

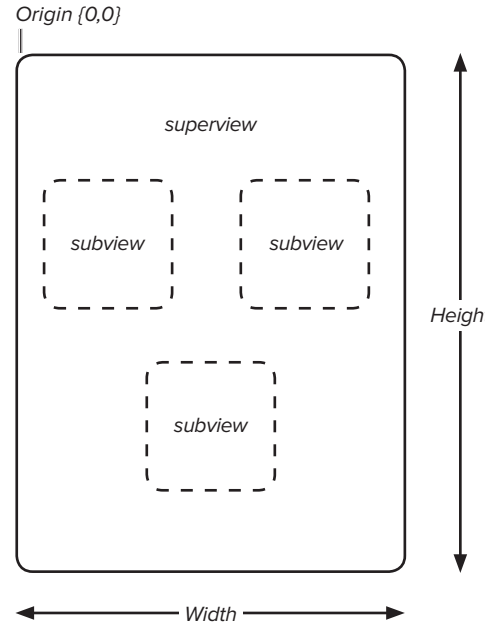
# Views

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*.

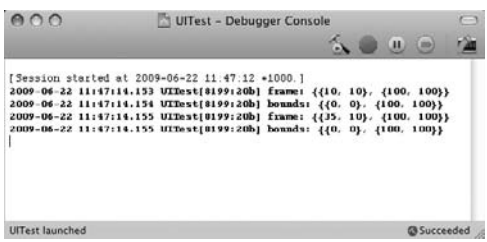



**A** Child views (subviews) are nested inside their parent view (superview). A view's origin is at the top-left corner.

## 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**.


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




C 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.

```
- (void)viewDidLoad {

    CGRect viewFrame = CGRectMake(10,10,100,100);
    UIView *myView = [[UIView alloc] initWithFrame:
                      viewFrame];

    myView.backgroundColor = [UIColor blueColor];

    [self.view addSubview:myView];
    [myView release];

}
```

## To add a view to your application:

1. 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.

2. 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.

**TIP 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 C: 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.

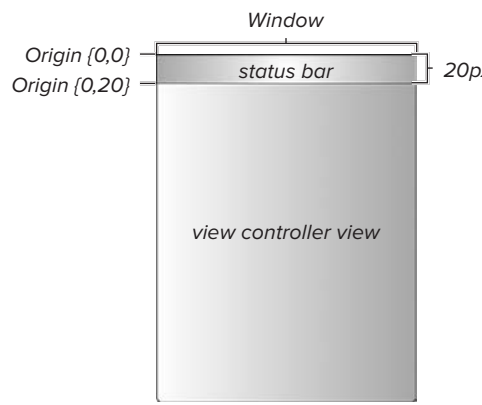
If you tried to run this code in your application, you'd see a gap at the top of the subview **D**.

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 **E**. 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.

**TIP** **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()`.**


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




**E** 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
→ (frameCenter.x-width,
→ frameCenter.y-height,width*2,
→ 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.

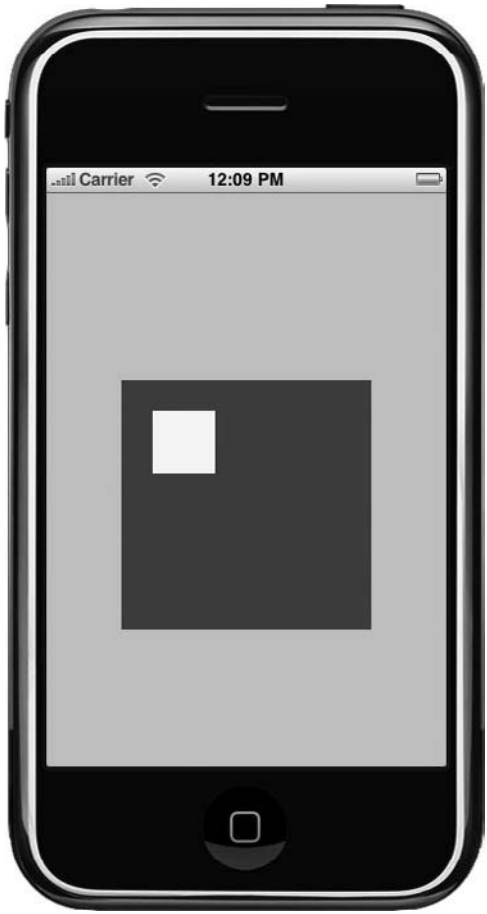
You should see the view grow in size over a period of one second. **Code Listing 4.3** shows the completed code.

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

**TIP 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];
}
```




F 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:

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

```
CGRect subViewFrame =
→ CGRectInset(myView.bounds,
→ 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 F. 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*

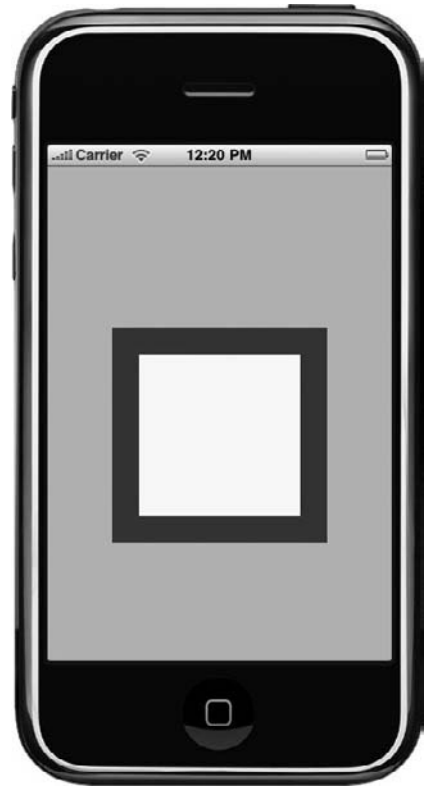
**Code Listing 4.4** shows this code updated to use an autoresizing mask. Notice how you set all four margins of the subview using the bitwise OR operator (the | symbol) between the constant values (**Table 4.1**). Notice also that even though the animation is specified on the superview, the subview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

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

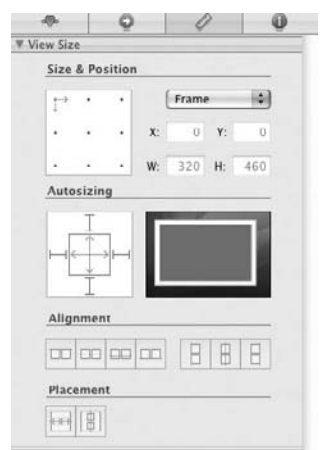
## Custom drawing

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. |

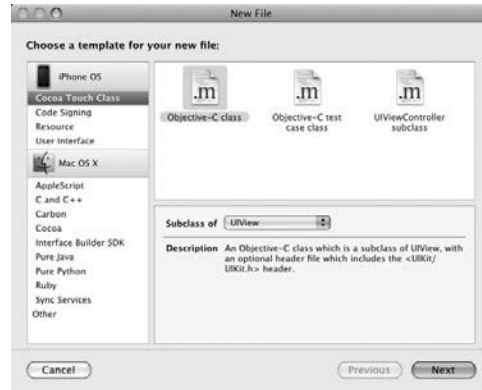
## To create a custom rounded-corner view:

1. In Xcode, select File > New File. Create a new Objective-C class, making sure that "Subclass of" is set to UIView ❶. Save the file as roundedCornerView.
2. Open roundedCornerView.m, and modify your code to look like **Code Listing 4.5**.
3. 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.
4. Build and run your application to see the result with rounded corners for the views ❷.

   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.
5. 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*




❶ Adding a custom class to draw the rounded corner view.


❷ 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];
}


@end
```

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.

   ```
   CGContextRef ctx =
   → UIGraphicsGetCurrentContext();

   CGContextSetStrokeColorWithColor
   → (ctx, parentColor.CGColor);

   CGContextSetLineWidth(ctx,
   → lineWidth);
   ```

7. Finally, call a custom function that draws 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 =
→ 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 =
→ CGAffineTransformMakeRotation
→ (radiansForDegrees(180));

CGAffineTransform myTransform =
→ CGAffineTransformConcat
→ (scale,rotate);

myView.transform = myTransform;
```

   Note how you can combine transformations using the `CGAffineTransformConcat()` function.

   **Code Listing 4.7** shows the completed code.

2. Build and run your application K.

   Your view should rotate and scale at the same time.


K The view both rotating and scaling.

**TIP** **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.**

**TIP** **You can return a view to its original state by setting its `transform` property to `CGAffineTransformIdentity`.**

**Code Listing 4.7** Rotating and scaling the view.

```
CGFloat radiansForDegrees(CGFloat degrees)
{
    return (M_PI * degrees / 180.0);
}

- (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];
}
```

# 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]
→ 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 **B**:

```
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.

```
- (void)viewDidLoad {

    UIImage *anImage = [UIImage imageNamed:
                        @"myImage.png"];
    UIImageView *myImageView = [[UIImageView alloc]
                                initWithImage:anImage];

    [self.view addSubview:myImageView];
    [myImageView release];
}
```


**A** The image displaying a graphic.


**B** Resizing the image view.


C Resizing the image view while maintaining its aspect ratio.

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 C. 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.

**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];
}
```

### 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];
```

3. You can control the speed of the animation (in seconds) and number of times the animation is repeated. The default is `0`, making the animation loop indefinitely:

```
[myImageView
→setAnimationDuration:0.5];

[myImageView setAnimation
→RepeatCount:0];
```

4. To begin the animation, add the following:

```
[myImageView startAnimating];
```

5. To stop the animation, you call `stopAnimating`.

**TIP** **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 low-memory 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
→(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 =
→[[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];
}
```

3. 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 Ⓐ. You can hide these scroll bars using the **showsHorizontalScrollIndicator** and **showsVerticalScrollIndicator** properties.


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

**TIP** **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 **minimumZoomScale** 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.

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>
   ```

2. 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 **viewForZoomingInScrollView:** 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:

1. 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.contentSize =
→ CGSizeMake(scrollViewRect.size.
→ width * pageCount,1);
myScrollView.showsHorizontal
→ ScrollIndicator = NO;
myScrollView.showsVertical
→ ScrollIndicator = NO;
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.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:)
→orControlEvents:UIControlEvent
→ValueChanged];
```

4. Call the **createPages** method by adding three **UIView**s 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];

    //cleanup
    [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:myScrollView];
    [self.view addSubview:myPageControl];

    //cleanup
    [myPageControl release];
    [myScrollView release];
}

- (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];
}
```

# 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 you can 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.

```
- (void)viewDidLoad {

    CGRect labelFrame = CGRectMake(10,10,200,44);

    UILabel *myLabel = [[UILabel alloc] initWithFrame:
                        labelFrame];

    myLabel.backgroundColor = [UIColor clearColor];
    myLabel.textColor = [UIColor redColor];
    myLabel.font = [UIFont systemFontOfSize:18.0];
    myLabel.text = @"Hello World!";

    [self.view addSubview:myLabel];
    [myLabel release];
}
```

**TABLE 4.2 Fonts available on the iPhone**

| 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 |

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**).

**TIP** **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.

```
- (void)viewDidLoad {

    CGRect labelFrame = CGRectMake(10,10,200,44);

    UILabel *myLabel = [[UILabel alloc] initWithFrame:
                        labelFrame];

    myLabel.backgroundColor = [UIColor clearColor];
    myLabel.textColor = [UIColor redColor];
    myLabel.font = [UIFont fontWithName:@"Verdana"
                                   size:18.0];
    myLabel.numberOfLines = 2;
    myLabel.text = @"Hello World!\nSecond line";

    myLabel.shadowColor = [UIColor darkGrayColor];
    myLabel.shadowOffset = CGSizeMake(1.0,1.0);

    [self.view addSubview:myLabel];
    [myLabel release];
}
```


Ⓐ 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]
→initWithProgressViewStyle:
→UIProgressViewStyleDefault];
```

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

```
[myProgressView setProgress:0.33];
```

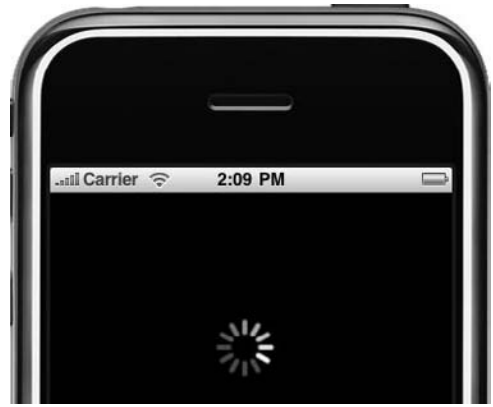
Although you set the frame of the progress view, the maximum height of a progress view is 9 pixels, so any larger value will be ignored.

**Code Listing 4.15** shows an example of using **`UIProgressView`** with the progress updated in a timer to simulate a long-running task.

**TIP 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
                                              target: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.

```
- (void)viewDidLoad {

    [super viewDidLoad];

    [self.view setBackgroundColor:[UIColor blackColor]];

    UIActivityIndicatorView *myActivityView = [[UIActivityIndicatorView alloc]
                                               initWithActivityIndicatorStyle:UIActivityIndicatorViewStyleWhiteLarge];

    CGRect activityFrame = CGRectMake(130,100,50,50);
    [myActivityView setFrame:activityFrame];

    [myActivityView startAnimating];

    [self.view addSubview:myActivityView];

    [myActivityView release];
}
```

# 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 Ⓐ:

```
UIAlertView *myAlert =
→[[UIAlertView alloc]
→initWithTitle:@"title"
→message:@"message"
→delegate:nil
→cancelButtonTitle:@"OK",
→otherButtonTitles:nil]
→[myAlert show];
```

2. Using the **`otherButtonTitles`** property, you can create the same alert view with up to four additional buttons Ⓑ:

```
UIAlertView *myAlert =
→[[UIAlertView alloc]
→initWithTitle:@"title"
→message:@"message"
→delegate:nil
→cancelButtonTitle:@"OK"
→otherButtonTitles:@"button1",
→@"button2", @"button3",
→@"button4",nil];
```


Ⓐ A bare-bones alert view.


Ⓑ An alert view with several buttons added.

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.

3. You can add buttons after creating your alert view by using the **`addButtonWithTitle:`** method:

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

4. 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.

**TIP Using the `dismissWithClickedButtonIndex: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.**

**TIP 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.**

**Code Listing 4.17** Display an alert view.

```
- (void)viewDidLoad {

    [super viewDidLoad];

    UIAlertView *myAlert = [[UIAlertView alloc] initWithTitle:@"title"
                                                      message:@"message"
                                                     delegate:self
                                            cancelButtonTitle:@"OK"
                                            otherButtonTitles:nil];

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

    [myAlert show];

    [myAlert release];
}

- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex {

    NSLog(@"you clicked button: %i",buttonIndex);
}
```

## Confirming an action

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

## To create an action sheet:

1. An action sheet is created in a similar way to an alert view ❻:

```
UIActionSheet *mySheet =
→[[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.

3. 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.


❻ 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];


    mySheet.actionSheetStyle = UIActionSheetStyleBlackOpaque;
    [mySheet showInView:self.view];

    [mySheet release];
}

- (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);

}
```

## 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 Ⓐ.

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).


Ⓐ A picker view being used to select a date.

## To create a simple picker view:

1. Add the protocol declarations to your interface definition:

```
@interface UITestViewController :
→ UIViewController
→ <UIPickerViewDataSource,
→ 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];
```

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];
    myPicker.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 Ⓑ:

```
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:

1. 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:numberofRowsInComponent:`** 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, **`pickerView:widthFormComponent:`**, and set the widths of the components to different values:

```
if (component == 0)
  return 100.0;
else
  return 200.0;
```


Ⓑ The picker view shows the sample data.


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

3. 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.
4. 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
{
    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];

}
```

*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 Ⓓ.

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:


Ⓓ 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.

```
-(void)pickerChanged:(id)sender {

    NSLog(@"value: %@",[sender date]);
}

- (void)viewDidLoad {

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

    UIDatePicker *myPicker = [[UIDatePicker alloc]
                              initWithFrame:pickerFrame];

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

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

# 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 Ⓐ.

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

## To create a toolbar:

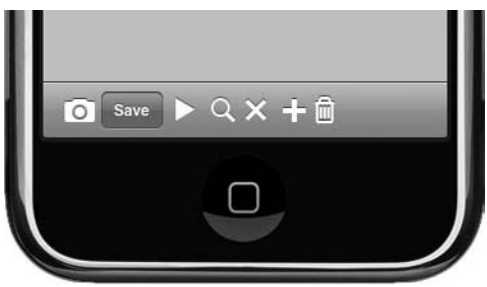
1. As with many other views, you can create a toolbar with the **initWithFrame:** method. Use the size of the main view to calculate the y position of the toolbar. This is important since you may not know the orientation of the iPhone and want the toolbar to sit at the bottom of the screen.

```
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 =
→UIViewAutoresizingFlexible
→Width | UIViewAutoresizing
→FlexibleLeftMargin | UIView
→AutoresizingFlexibleRight
→Margin | UIViewAutoresizing
→FlexibleTopMargin;
```


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




C Some of the available system item styles.

3. 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 **`initWithBarButtonSystemItem: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 C. 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 ❶.


❶ 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]);
}

- (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
                                 |
                                 UIViewAutoresizingFlexibleRightMargin
                                 |
                                 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
                                                               target: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];
}
```

# 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:

1. 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.
2. As with most other views, you use the **`initWithFrame:`** method to create them:

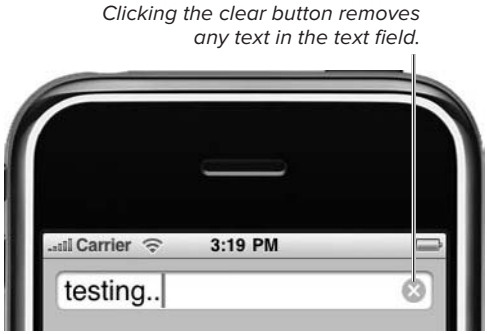
   ```
   CGRect textRect = CGRectMake
   → (10,10,300,20);

   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 Ⓐ.

   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.


Ⓐ Border styles available for text fields.




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;

myTextField.clearButtonMode =
→UITextFieldViewModeWhileEditing;
```

   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.
2. You can choose from a number of keyboard styles C, 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*

4. 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.
6. 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.
8. 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 **textFieldShouldReturn:** 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];
}
```

9. 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];
```

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

**TIP 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:shouldChangeCharactersInRange: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.

```
- (BOOL)textField:(UITextField *)textField
shouldChangeCharactersInRange:(NSRange)range
replacementString:(NSString *)string
{
    //limit text field to 10 chars
    int MAX_CHARS = 10;
    NSMutableString *newText = [NSMutableString
                                stringWithString:textField.text];
    [newText replaceCharactersInRange:range
                           withString:string];
    return ([newText length] <= MAX_CHARS);
}
```

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.

**Code Listing 4.25** Restricting the contents of a text field to numeric values.

```
- (BOOL)textField:(UITextField *)textField
shouldChangeCharactersInRange:(NSRange)range
replacementString:(NSString *)string
{
    //limit text field to numeric values only
    NSCharacterSet *numberSet = [NSCharacterSet
                                 decimalDigitCharacterSet];
    for (NSUInteger i=0; i<[string length]; i++)
    {
        unichar ch = [string characterAtIndex:i];
        if (![numberSet characterIsMember:ch])
            return NO;
    }
    return YES;
}
```

## 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.


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: **`UIDataDetectorTypePhoneNumber`** for phone numbers and **`UIDataDetectorTypeLink`** for Web **`http:`** links. To enable both, set the **`dataDetectorTypes`** 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 D, 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.

4. 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 *myScrollView;
UITextView *myTextView;

@implementation UITestViewController

-(void)buttonClick:(id)sender
{
    [myTextView resignFirstResponder];
}

- (void)viewDidLoad {

    CGRect viewRect = [self.view bounds];
    myScrollView = [[UIScrollView alloc] initWithFrame:viewRect];
    myScrollView.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:

1. 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.


A A web view displaying the Google homepage.

2. 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.

3. 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.

```
UIActivityIndicatorView *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;

    myWebView.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 **`stringByEvaluatingJavaScriptFromString:`** 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 =
→[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 ❶.

2. 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.

   *or*


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

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

3. 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.

**TIP** **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 = NO;
  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;
}
```

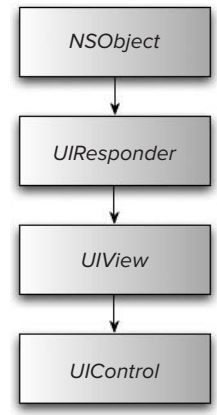
# Controls

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`** Ⓐ; this is how controls know how to draw themselves.

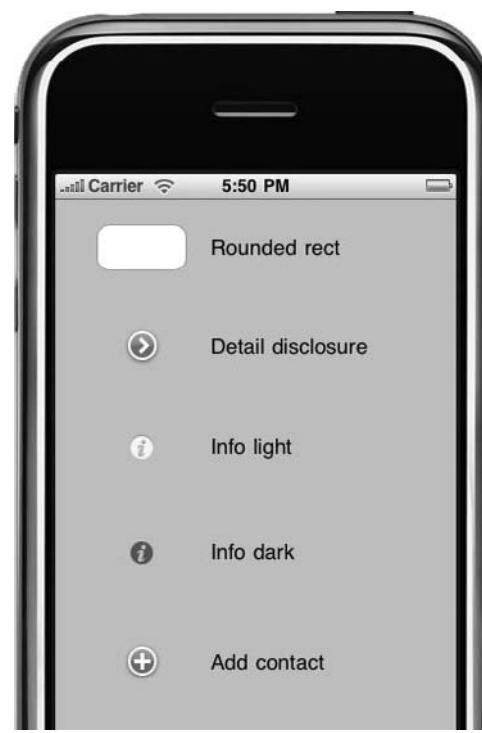
You'll never actually create instances of **`UIControl`** directly the way you do with **`UIView`**. **`UIControl`** is simply used to define a common set of functionality and behavior for its subclasses.

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.




Ⓐ The **`UIControl`** class hierarchy.




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

## Buttons

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

```
UIButton *myButton =
→[[UIButton buttonWithType:
→UIButtonTypeRoundedRect];
```

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:

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

   ```
   UIImage *buttonImage =
   → [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:
   → @selector(checkboxClick:)
   → forControlEvents:UIControlEvent
   → TouchUpInside];
   ```

3. 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.

**TIP** **If you specify an `image` or `title` for any button type other than `UIButtonTypeRoundedRect`, the button effectively becomes a button of `UIButtonTypeCustom`.**

## Switches

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

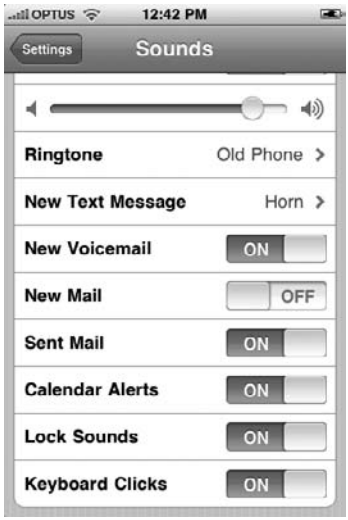
## To create a switch:

1. Use the `initWithFrame:` method:

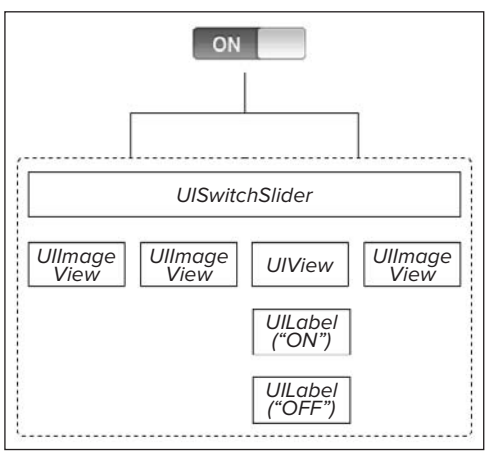
   ```
   CGRect switchRect = CGRectMake
   → (120,50,0,0);

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

**Code Listing 4.29** Creating a check box.

```
- (void)checkboxClick:(UIButton *)sender {

    sender.selected = ! sender.selected;
}

- (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];

}
```




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.

2. When you change a switch's value, it generates a **`UIControlEventValueChanged`** event:

```
[mySwitch addTarget:self action:
→ @selector(switchAction:)
→ forControlEvents:UIControlEvent
→ ValueChanged];
```

3. 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 hierarchy 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:

1. To retrieve the two **`UILabel`**s 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*

2. 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:

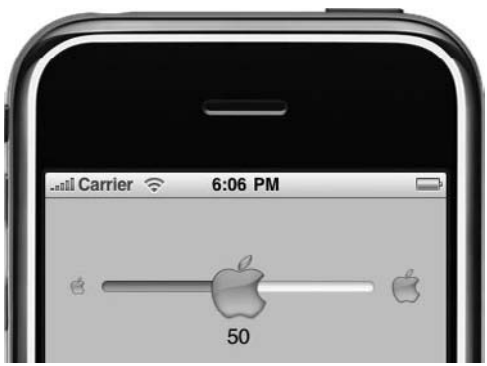
```
onLabel.text = @"YES";

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");
}

- (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.




F 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 E.

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 F.

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.

**TIP 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.**

**TIP The `stretchableImageWithLeftCapWidth: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 *lblSliderValue;

@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];
}


- (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];
    UIImage *thumbImage      = [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];
}
```


G A segment control.

## Segmented controls

The **`UISegmentedControl`** consists of a horizontal control divided into segments G. 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:

1. Create an array of **`UIImage`**s or **`NSString`**s, 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.

```
- (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];

    [self.view addSubview:mySegment];
    [arrSegments release];
    [mySegment release];
}
```

3. 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.

4. Select segments using the **`setSelectedSegmentIndex:`** method.

5. 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 **`removeSegmentsAtIndex: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 ❽. 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.


❽ The 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



## A

## B



## C

## D

## E



## F

## G



## H



## I

## J



## K

## L



## M

## N



## O

## P



## Q



## R

## S

## T

## U



## V

## W



## X

## Z