John Ray

In Full Color

Figures and code appear as they do in Xcode 4.2+

Covers iOS 5, Xcode 4.2+, Storyboards, iPhone, iPad, and More!

Additional files and updates available online

Sams Teach Yourself

iOS® 5 Application Development

in 24 Hours

SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

John Ray

Sams Teach Yourself

iOS® 5 Application Development

in 24 Hours

SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA

Sams Teach Yourself iOS® 5 Application Development in 24 Hours

ISBN-13: 978-0-672-33576-1
ISBN-10: 0-672-33576-X

Library of Congress Cataloging-in-Publication Data is on file.

Printed in the United States of America

Second Printing: September 2012

Trademarks

Warning and Disclaimer

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales
international@pearson.com

Associate Publisher
Greg Wiegand

Acquisitions Editor
Laura Norman

Development Editor
Keith Cline

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Copy Editor
Keith Cline

Indexer
Larry Sweazy

Proofreader
Karen Gill

Technical Editor
Anne Groves

Publishing Coordinator
Cindy Teeters

Designer
Gary Adair

Compositor
Nonie Ratcliff

Contents at a Glance

Table of Contents

About the Author

John Ray is currently serving as a Senior Business Analyst and Development Team Manager for the Ohio State University Research Foundation. He has written numerous books for Macmillan/Sams/Que, including *Using TCP/IP: Special Edition*, *Teach Yourself Dreamweaver MX in 21 Days*, *Mac OS X Unleashed*, and *Teach Yourself iPad Development in 24 Hours*. As a Macintosh user since 1984, he strives to ensure that each project presents the Macintosh with the equality and depth it deserves. Even technical titles such as *Using TCP/IP* contain extensive information about the Macintosh and its applications and have garnered numerous positive reviews for their straightforward approach and accessibility to beginner and intermediate users.

You can visit his website at http://teachyourselfios.com or follow him on Twitter at #iOSIn24.

Dedication

To the crazy ones.
Thank you, Steve Jobs.

Acknowledgments

Thank you to the group at Sams Publishing—Laura Norman, Keith Cline, Anne Groves—for not giving up on this book, despite the changes, delays, and other challenges that we encountered along the way. I'm not sure how you manage to keep all of the files, figures, and information straight, but on this end it looks like magic.

As always, thanks to my family and friends for feeding me and poking me with a stick to keep me going.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone or email address. I will carefully review your comments and share them with the author and editors who worked on the book.

E-mail: feedback@quepublishing.com

Mail: Greg Wiegand
Associate Publisher
Sams Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

Introduction

In less than half a decade, the iOS platform has changed the way that we, the public, think about our mobile computing devices. Only a few years ago, we were thrilled by phones with postage-stamp-sized screens, tinny audio, built-in tip calculators, and text-based web browsing. Times have indeed changed. With full-featured applications, an interface architecture that demonstrates that small screens can be effective workspaces, and touch controls unrivaled on any platform, the iPhone brings us the convenience of desktop computing within our pockets.

When Steve Jobs introduced the iPad, people laughed at the name and the idea that "a big iPod Touch" could be magical. In the 2 years that have passed since its introduction, the iPad has become the de facto standard for tablet computing and shows no signs of slowing down. Rarely a week goes by when I don't read a review of a new app that is described as "magical" and that could only have been created on the iPad. The excitement and innovation surrounding iOS and the sheer enjoyment of using the iOS devices has led it to become the mobile platform of choice for users and developers alike.

With Apple, the user experience is key. The iOS is designed to be controlled with your fingers rather than by using a stylus or keypad. The applications are "natural" and fun to use, instead of looking and behaving like a clumsy port of a desktop app. Everything from interface to application performance and battery life has been considered. The same cannot be said for the competition.

Through the App Store, Apple has created the ultimate digital distribution system for developers. Programmers of any age or affiliation can submit their applications to the App Store for just the cost of a modest yearly Developer Membership fee. Games, utilities, and full-feature applications have been built for everything from pre-K education to retirement living. No matter what the content, with a user base as large as the iPhone, iPod Touch, and iPad, an audience exists.

Each year, Apple introduces new devices—bringing larger, faster, and higher-resolution capabilities to the iOS family. With each new hardware refresh come new development opportunities and new ways to explore the boundaries between software and art.

My hope is that this book will bring iOS development to a new generation of developers. *Teach Yourself iOS 5 Development in 24 Hours* provides a clear natural progression of skills development, from installing developer tools and registering your device with Apple, to submitting an application to the App Store. It's everything you need to get started in 24 one-hour lessons.

Who Can Become an iOS Developer?

If you have an interest in learning, time to invest in exploring and practicing with Apple's developer tools, and an Intel Macintosh computer running Lion, you have everything you need to begin creating software for iOS.

Developing an app won't happen overnight, but with dedication and practice, you can be writing your first applications in a matter of days. The more time you spend working with the Apple developer tools, the more opportunities you'll discover for creating new and exciting projects.

You should approach iOS application development as creating software that *you* want to use, not what you think others want. If you're solely interested in getting rich quick, you're likely to be disappointed. (The App Store is a crowded marketplace—albeit one with a lot of room—and competition for top sales is fierce.) However, if you focus on building apps that are useful and unique, you're much more likely to find an appreciative audience.

Who Should Use This Book?

This book targets individuals who are new to development for the iPhone and iPad and have experience *using* the Macintosh platform. No previous experience with Objective-C, Cocoa, or the Apple developer tools is required. Of course, if you do have development experience, some of the tools and techniques may be easier to master, but the authors do not assume that you've coded before.

That said, some things are expected of you, the reader. Specifically, you must be willing to invest in the learning process. If you just read each hour's lesson without working through the tutorials, you will likely miss some fundamental concepts. In addition, you need to spend time reading the Apple developer documentation and researching the topics presented in this book. There is a vast amount of information on iOS development available, and only limited space in this book. This book covers what you need to forge your own path forward.

What Is (and Isn't) in This Book?

The material in this book specifically targets iOS release 5 and later on Xcode 4.2 and later. Much of what you'll be learning is common to all the iOS releases, but this book also covers several important areas that have only come about in iOS 4 and 5, such as gesture recognizers, embedded video playback with AirPlay, Core Image, multitasking, universal (iPhone/iPad) applications, and more!

Unfortunately, this is not a complete reference for the iOS APIs; some topics just require much more space than this book allows. Thankfully, the Apple developer documentation is available directly within the free tools you'll be installing in Hour 1, "Preparing Your System and iDevice for Development." In many hours, you'll find a section titled "Further Exploration." This identifies additional related topics of interest. Again, a willingness to explore is an important quality in becoming a successful developer.

Each coding lesson is accompanied by project files that include everything you need to compile and test an example or, preferably, follow along and build the application yourself. Be sure to download the project files from the book's website at http://teachyourselfios.com. If you have issues with any projects, view the posts on this site to see whether a solution has been posted.

In addition to the support website, you can follow along on Twitter! Search for #iOSIn24 on Twitter to receive official updates and tweets from other readers. Use the hashtag #iOSIn24 in your tweets to join the conversation. To send me messages via Twitter, begin each tweet with @johnemeryray.

This page intentionally left blank

HOUR 5

Exploring Xcode's Interface Builder

What You'll Learn in This Hour:

- Where Xcode's Interface Builder fits in the development process
- The role of storyboards and scenes
- How to build a user interface using the Object Library
- Common attributes that can be used to customize interface elements
- Ways to make your interface accessible to the visually impaired
- How to link interfaces to code with outlets and actions

Over the past few hours, you've become familiar with the core iOS technologies, Xcode projects, and iOS Simulator. Although these are certainly important skills for becoming a successful developer, there's nothing quite like laying out your first iOS application interface and watching it come to life in your hands.

This hour introduces you to Interface Builder: the remarkable user interface editor integrated into Xcode. Interface Builder provides a visual approach to application interface design that is fun, intuitive, and deceptively powerful.

Understanding Interface Builder

Let's get it out of the way up front: Yes, Interface Builder (or IB for short) does help you create interfaces for your applications, but it isn't a just a drawing tool for GUIs; it helps you symbolically build application functionality without writing code. This translates to fewer bugs, less development time, and easier-to-maintain projects.

If you read through Apple's developer documentation, you'll see Interface Builder referred to as an "editor" within Xcode. This is a bit of an oversimplification of a tool that previously existed as a standalone application in the Apple Developer Suite. An understanding of IB and its use is as fundamentally important to iOS development as Objective-C. Without Interface Builder, creating the most basic interactive applications would be an exercise in frustration.

This hour focuses on navigating Interface Builder and will be key to your success in the rest of the book. In Hour 6, "Model-View-Controller Application Design," you combine what you've learned about Xcode projects, the code editor, Interface Builder, and iOS Simulator for the first time. So, stay alert and keep reading.

The Interface Builder Approach

Using Xcode and the Cocoa toolset, you can program iOS interfaces by hand—instantiating interface objects, defining where they appear on the screen, setting any attributes for the object, and finally, making them visible. For example, in Hour 2, "Introduction to Xcode and the iOS Simulator," you entered this listing into Xcode to make your iDevice display the text `Hello Xcode` in the corner of the screen:

```
UILabel *myMessage;
myMessage=[[UILabel alloc]
           initWithFrame:CGRectMake(30.0,50.0,300.0,50.0)];
myMessage.font=[UIFont systemFontOfSize:48];
myMessage.text=@"Hello Xcode";
myMessage.textColor = [UIColor colorWithPatternImage:
                       [UIImage imageNamed:@"Background.png"]];
[self.window addSubview:myMessage];
```

Imagine how long it would take to build interfaces with text, buttons, images, and dozens of other controls, and think of all the code you'd need to wade through just to make small changes.

Over the years, there have been many different approaches to graphical interface builders. One of the most common implementations is to enable the user to "draw" an interface but, behind the scenes, create all the code that generates that interface. Any tweaks require the code to be edited by hand (hardly an acceptable situation).

Another tactic is to maintain the interface definition symbolically but attach the code that implements functionality directly to interface elements. This, unfortunately, means that if you want to change your interface or swap functionality from one UI element to another, you have to move the code as well.

Interface Builder works differently. Instead of autogenerating interface code or tying source listings directly to interface elements, IB builds live objects that connect to your application code through simple links called *connections*. Want to change how

a feature of your app is triggered? Just change the connection. As you'll learn a bit later this hour, changing how your application works with the objects you create in Interface Builder is, quite literally, a matter of connecting or reconnecting the dots as you see fit.

The Anatomy of an Interface Builder Storyboard

Your work in Interface Builder results in an XML file called a *storyboard*, containing a hierarchy of objects for each unique screen that your application is going to display. The objects could be interface elements—buttons, toggle switches, and so forth—but might also be other noninterface objects that you will need to use. The collection of objects for a specific display is called a *scene*. Storyboards can hold as many scenes as you need, and even link them together visually via *segues*.

For example, a simple recipe application might have one scene that consists of a list of recipes the user can choose from. A second scene may contain the details for making a selected recipe. The recipe list could be set to segue to the detail view with a fancy fade-out/fade-in effect when the name of a recipe is touched. All of this functionality can be described visually in an application's storyboard file.

Storyboards aren't just about cool visuals, however. They also help you create usable objects without having to allocate or initialize them manually. When a scene in a storyboard file is loaded by your application, the objects described in it are instantiated and can be accessed by your code.

Instantiation, just as a quick refresher, is the process of creating an instance of an object that you can work with in your program. An instantiated object gains all the functionality described by its class. Buttons, for example, automatically highlight when clicked, content views scroll, and so on.

By the Way

The Storyboard Document Outline

What do storyboard files look like in IB? Open the Hour 5 Projects folder and double-click the file Empty.storyboard to open Interface Builder and display a barebones storyboard file. The contents of the file are shown visually in the IB Editor area, and hierarchically by scene in the Document Outline area located in the column to the left of the Editor area (see Figure 5.1).

If you do not see the Document Outline area in your Xcode workspace, choose Editor, Show Document Outline from the menu bar. You can also click the disclosure arrow in the lower-left corner of the Xcode Editor area.

By the Way

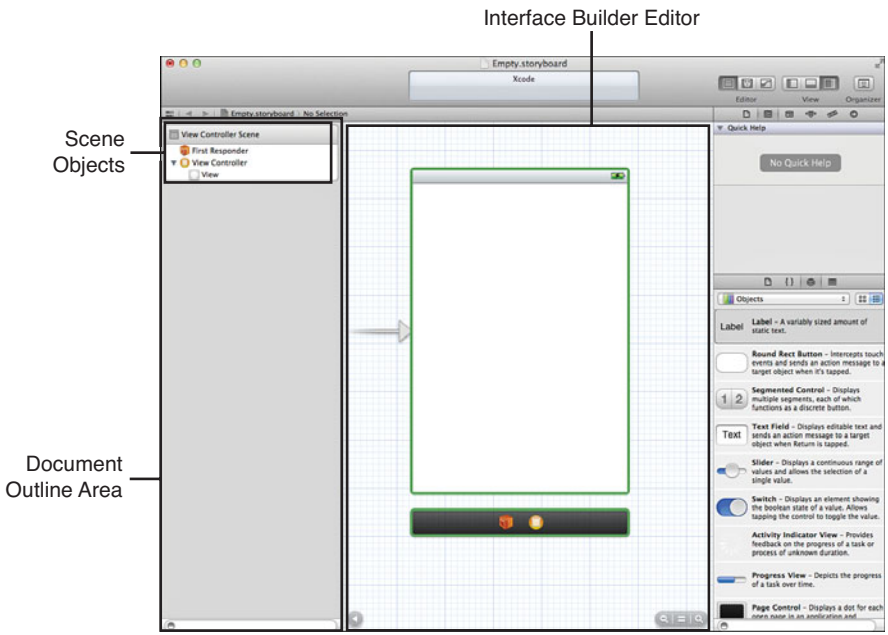

FIGURE 5.1
A storyboard scene's objects are represented by icons.

Note that there is only a single scene in the file: view controller scene. Single-scene storyboards will be the starting place for much of your interface work in this book because they provide plenty of room for collecting user input and displaying output. We explore multi-scene storyboards beginning in Hour 11, "Implementing Multiple Scenes and Popovers."

Three icons are visible in the view controller scene: First Responder, View Controller, and View. The first two are special icons used to represent unique noninterface objects in our application; these will be present in all storyboard scenes that you work with:

First Responder: The first responder stands for the object that the user is currently interacting with. When a user works with an iOS application, multiple objects could potentially respond to the various gestures or keystrokes that the user creates. The first responder is the object currently in control and interacting with the user. A text field that the user is typing into, for example, would be the first responder until the user moves to another field or control.

View Controller: The View Controller denotes the object that loads and interacts with a storyboard scene in your running application. This is the object that effectively instantiates all the other objects described within a scene. You'll learn more about the relationship between user interfaces and view controllers in Hour 6.

View: The View icon is an instance of the object `UIView` and represents the visual layout that will be loaded by the view controller and displayed on the iOS device's screen. Views are hierarchical in nature. This means that as you add controls to your interface they will be contained within the view. You can even add views *within* views to cluster controls or create visual elements that can be shown or hidden as a group.

By the Way

The storyboard shown in this example is about as "vanilla" as you can get. In larger applications with multiple scenes, you may want to either name your view controller class to better describe *what* it is actually controlling or set a descriptive label, such as Recipe Listing.

Using unique view controller names/labels also benefits the naming of scenes. Interface Builder automatically sets scene names to the name of the view controller or its label (if one is set) plus the suffix *scene*. If you label your view controller as Recipe Listing, for example, the scene name changes to Recipe Listing Scene. We'll worry about multiple scenes later in the book; for now, our projects will contain a generic class called View Controller that will be in charge of interacting with our single view controller scene.

As you build your user interfaces, the list of objects within your scenes will grow accordingly. Some user interfaces may consist of dozens of different objects, leading to rather busy and complex scenes, as demonstrated in Figure 5.2.

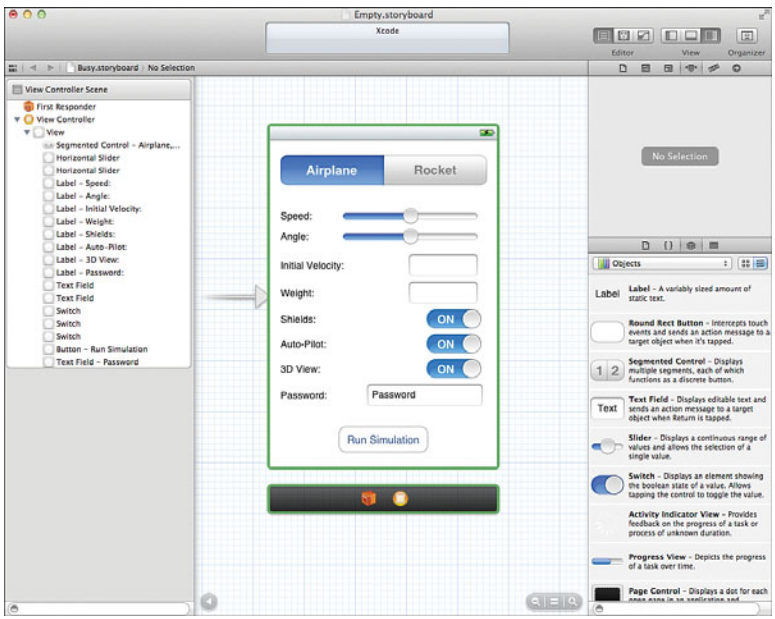

FIGURE 5.2
Storyboard scenes and their associated views can grow quite large and complex.

You can collapse or expand your hierarchy of views within the Document Outline area to help manage the information overload that you are bound to experience as your applications become more advanced.

Did You Know?

At its most basic level, a view (`UIView`) is a rectangular region that can contain content and respond to user events (touches and so forth). All the controls (buttons, fields, and so on) that you'll add to a view are, in fact, subclasses of `UIView`. This isn't necessarily something you need to be worried about, except that you'll be encountering documentation that refers to buttons and other interface elements referred to as *subviews* and the views that contain them as *superviews*.

Just keep in mind that pretty much everything you see onscreen can be considered a "view" and the terminology will seem a little less alien.

Working with the Document Outline Area Objects

The Document Outline area shows icons for objects in your application, but what good are they? Aside from presenting a nice list, do they provide any functionality?

Absolutely! Each icon gives you a visual means of referring to the objects they represent. You interact with the icons by dragging to and from them to create the connections that drive your application's features.

Consider an onscreen control, such as a button, that needs to be able to trigger an action in your code. By dragging from the button to the View Controller icon, you can create a connection from the GUI element to a method that you want it to activate. You can even drag from certain objects directly to your code, quickly inserting a variable or method that will interact with that object.

Xcode provides developers with a great deal of flexibility when working with objects in Interface Builder. You can interact with the actual UI elements in the IB Editor, or with the icons that represent them in the Document Outline area. In addition, any object that isn't directly visible in the user interface (such as the first responder and view controller objects) can be found in an icon bar directly below the user interface design in the Editor, as shown in Figure 5.3.

By the Way

If the icon bar below your view does not show any icons and is displaying the text *View Controller* instead, just click it. The icon bar frequently defaults to the name of a scene's view controller until it is clicked.

We go through a hands-on example later this hour so that you can get a feel for how interacting with and connecting objects works. Before we do that, however, let's look at how you go about turning a blank view into an interface masterpiece.

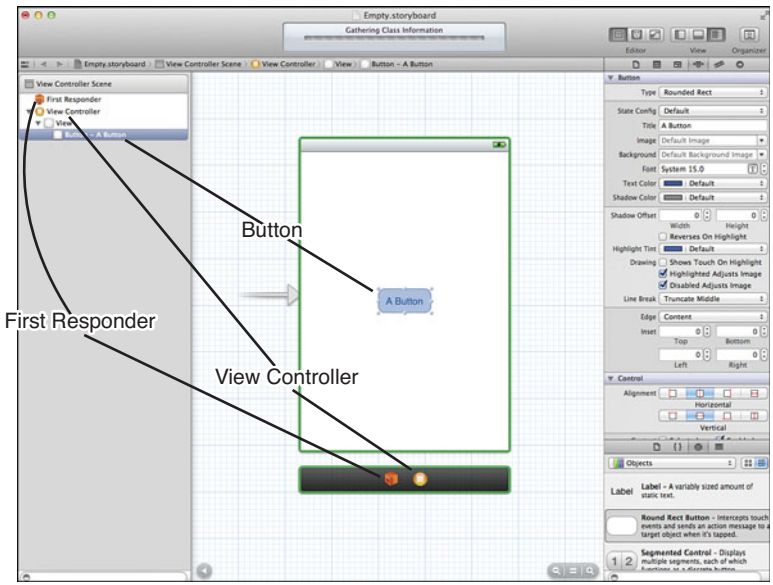

FIGURE 5.3
You will interact with objects either in the Editor or in the Document Outline area.

Creating User Interfaces

In Figures 5.1 and 5.2, you've seen an empty view and a fully fleshed-out interface. Now, how do we get from one to the other? In this section, we explore how interfaces are created with Interface Builder. In other words, it's time for the fun stuff.

If you haven't already, open the Empty.storyboard file included in this hour's Projects folder. Make sure the Document Outline area is visible and that the view can be seen in the Editor; you're ready to start designing an interface.

The Object Library

Everything that you add to a view, from buttons and images to web content, comes from the Object Library. You can view the Library by choosing View, Utilities, Show Object Library from the menu bar (Control+Option+Command+3). If it isn't already visible, the Utility area of the Xcode interface opens, and Object Library is displayed in the lower right. Make sure that the Objects item is selected in the pop-up menu at the top of the library so that all available options are visible.

Watch Out!

Libraries, Libraries, Everywhere!

Xcode has more than one library. The Object Library contains the UI elements you'll be adding in Interface Builder, but there are also File Template, Code Snippet, and Media libraries that can be activated by clicking the icons immediately above the Library area.

If you find yourself staring at a library that doesn't seem to show what you're expecting, click the cube icon above the library or reselect the Object Library from the menu to make sure you're in the right place.

When you click and hover over an element in the library, a popover is displayed with a description of how the object can be used in the interface, as shown in Figure 5.4. This provides a convenient way of exploring your UI options without having to open the Xcode documentation.

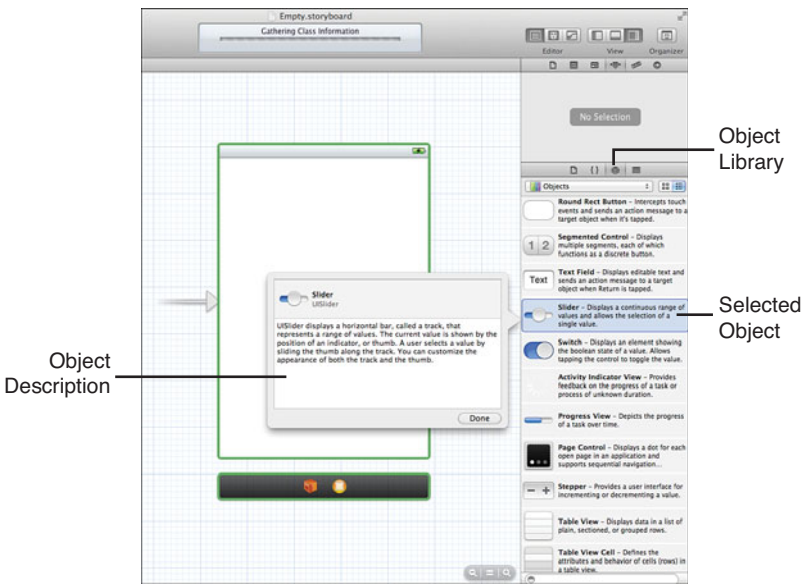

FIGURE 5.4
The library contains a palette of objects that can be added to your views.

Did You Know?

Using view buttons at the top of the library, you can switch between list and icon views of the available objects. You can also focus in on specific UI elements using the pop-up menu above the library listing. If you know the name of an object but can't locate it in the list, use the filter field at the bottom of the library to quickly find it.

Adding Objects to a View

To add an object to a view, just click and drag from the library to the view. For example, find the label object (`UILabel`) in the Object Library and drag it into the center of the view in the Editor. The label should appear in your view and read `Label`. Double-click the label and type **`Hello`**. The text will update, as shown in Figure 5.5, just as you would expect.

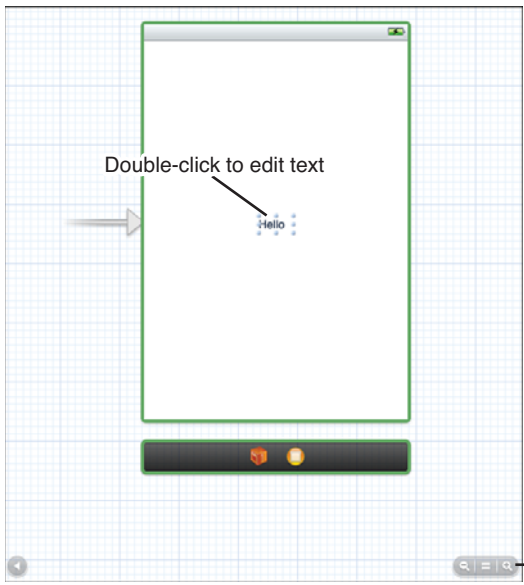

FIGURE 5.5
If an object contains text, in many cases, just double-click to edit it.

With that simple action, you've almost entirely replicated the functionality implemented by the code fragment earlier in the lesson. Try dragging other objects from the Object Library into the view (buttons, text fields, and so on). With few exceptions, the objects should appear and behave just the way you'd expect.

To remove an object from the view, click to select it, and then press the Delete key. You may also use the options under the Edit menu to copy and paste between views or duplicate an element several times within a view.

By the Way

The +/- magnifying glasses in the lower right of the Editor area will zoom in and out on your interface for fine-tuning a scene. This will be useful when creating storyboards with multiple scenes. Unfortunately, you cannot edit a scene when zoomed out, so Apple provides the = button to quickly jump back and forth between a 100% view and your last chosen zoom setting.

Working with the IB Layout Tools

Instead of relying on your visual acuity to position objects in a view, Apple has included some useful tools for fine-tuning your layout. If you've ever used a drawing program like OmniGraffle or Adobe Illustrator, you'll find many of these familiar.

Guides

As you drag objects in a view, you'll notice guides (shown in Figure 5.6) appearing to help with the layout. These blue, dotted lines will be displayed to align objects along the margins of the view, to the centers of other objects in the view, and to the baseline of the fonts used in the labels and object titles.

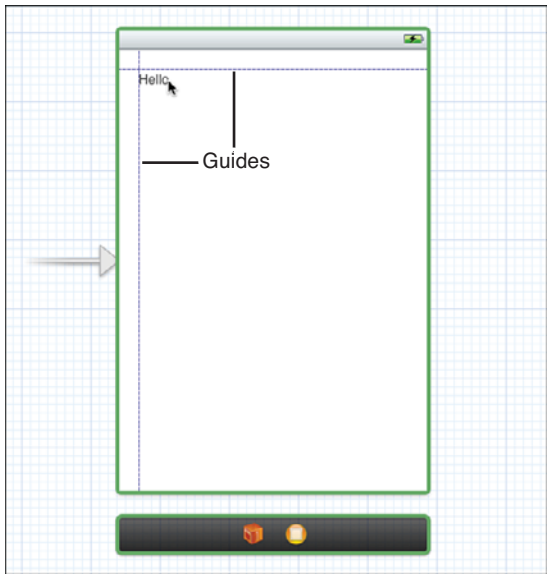

FIGURE 5.6
Guides help position your objects within a view.

As an added bonus, guides automatically appear to indicate the approximate spacing requirements of Apple's interface guidelines. If you're not sure why it's showing you a particular margin guide, it's likely that your object is in a position that Interface Builder considers "appropriate" for something of that type and size.

Did You Know?

You can manually add your own guides by choosing Editor, Add Horizontal Guide or by choosing Editor, Add Vertical Guide.

Selection Handles

In addition to the layout guides, most objects include selection handles to stretch an object horizontally, vertically, or both. Using the small boxes that appear alongside an object when it is selected, just click and drag to change its size, as demonstrated using a button in Figure 5.7.

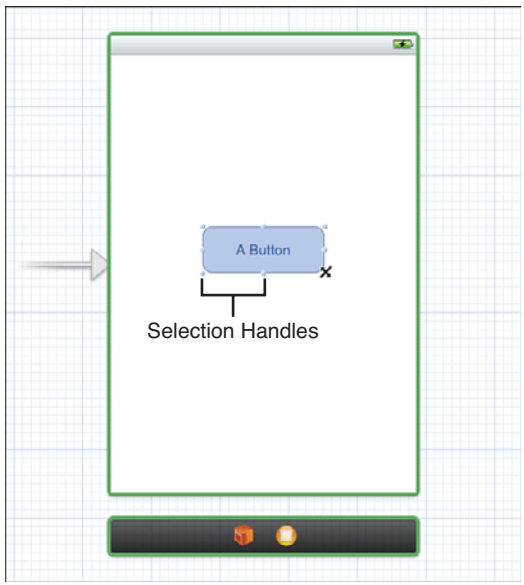

FIGURE 5.7
Use the resize handles around the perimeter of an object to change its size.

Note that some objects constrain how you can resize them; this preserves a level of consistency within iOS application interfaces.

Alignment

To quickly align several objects within a view, select them by clicking and dragging a selection rectangle around them or by holding down the Shift key, and then choose Editor, Align and an appropriate alignment type from the menu.

For example, try dragging several buttons into your view, placing them in a variety of different positions. To align them based on their horizontal center (a line that runs vertically through each button's center), select the buttons, and then choose Editor, Align, Horizontal Centers. Figure 5.8 shows the before and after results.

> To fine-tune an object's position within a view, select it, and then use the arrow keys to position it left, right, up, or down, 1 pixel at a time.

Did You Know?

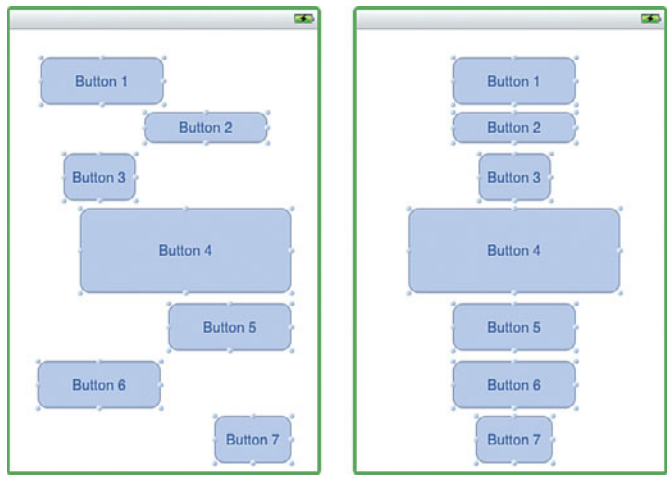

FIGURE 5.8
Use the Align menu to quickly align a group of items to an edge or center.

The Size Inspector

Another tool that you may want to use for controlling your layout is the Size Inspector. Interface Builder has a number of "inspectors" for examining the attributes of an object. As the name implies, the Size Inspector provides information about sizes, but also position and alignment. To open the Size Inspector, first select the object (or objects) that you want to work with, and then click the ruler icon at the top of the Utility area in Xcode. Alternatively, choose View, Utilities, Show Size Inspector or press Option+Command+5 (see Figure 5.9).

Using the fields at the top of the inspector, you can view or change the size and position of the object by changing the coordinates in the Height/Width and X/Y fields. You can also view the coordinates of a specific portion of an object by clicking one of the black dots in the size and grid to indicate where the reading should come from.

By the Way

Within the Size and Position settings, notice a drop-down menu where you can choose between Frame Rectangle and Layout Rectangle. These two settings will usually be similar, but there is a slight difference. The frame values represent the exact area an object occupies onscreen, whereas the layout values take into account spacing around the object.

The Autosizing settings of the Size Inspector determine how controls resize/reposition themselves when the device changes orientation. You'll learn more about these in Hour 16, "Building Rotatable and Resizable User Interfaces."

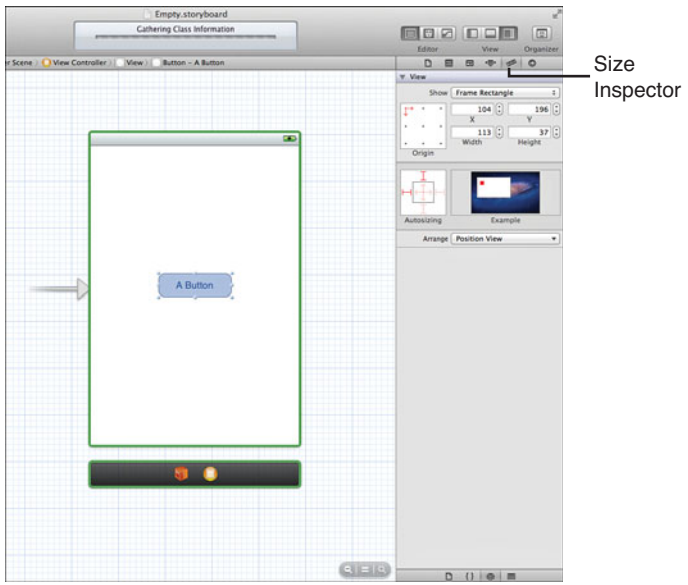

FIGURE 5.9
The Size Inspector enables you to adjust the size and position of one or more objects.

Finally, the same controls found under Editor, Align can be accessed via the pop-up menu at the bottom of the inspector. Choose your objects, and then choose an alignment from the menu.

> Hold down the option after selecting an object in Interface Builder. As you move your mouse around, it will show the distance between the selected object and other objects that you point to.

Did You Know?

Customizing the Interface Appearance

How your interface appears to the end user isn't just a combination of control sizes and positions. For many kinds of objects, literally dozens of different attributes can be adjusted. Although you could certainly configure things such as colors and fonts in your code, it's easier to just use the tools included in Interface Builder.

Using the Attributes Inspector

The most common place you'll tweak the way your interface objects appear is through the Attributes Inspector, available by clicking the slider icon at the top of the Utility area. You can also choose View, Utilities, Show Attributes Inspector (Option+Command+4) if the Utility area isn't currently visible. Let's run through a quick example to see how this works.

Make sure the Empty.storyboard file is still open and that you've added a text label to the view. Select the label, and then open the Attributes Inspector, shown in Figure 5.10.

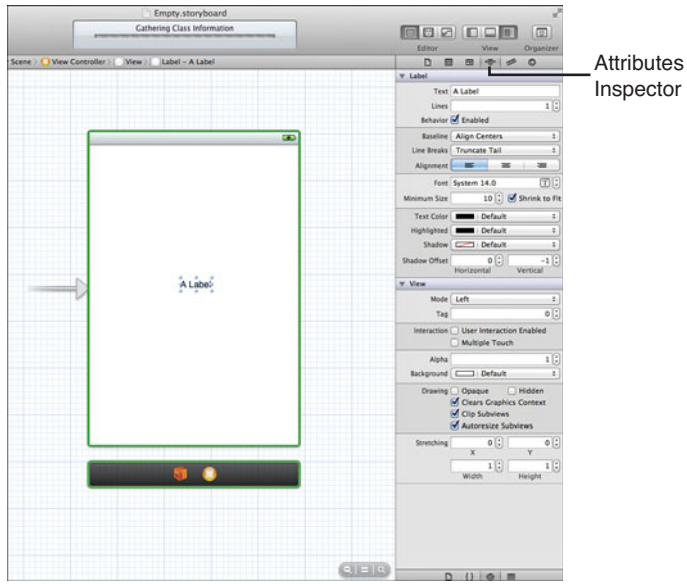

FIGURE 5.10
To change how an object looks and behaves, select it and then open the Attributes Inspector.

The top portion of the Attributes Inspector will contain attributes for the specific object. In the case of the text object, this includes settings such as font, size, color, and alignment (everything you'd expect to find for editing text).

In the lower portion of the inspector are additional inherited attributes. Remember that onscreen elements are a subclass of a view. Therefore, all the standard view attributes are also available for the object and for your tinkering enjoyment. In many cases, you'll want to leave these alone, but settings such as background and transparency can come in handy.

Did You Know?

Don't get hung up on trying to memorize every attribute for every control now. I cover interesting and important attributes when they are needed throughout the book.

Feel free to explore the many different options available in the Attributes Inspector to see what can be configured for different types of objects. There is a surprising amount of flexibility to be found within the tool.

The attributes you change in Interface Builder are simply properties of the object's class. To help identify what an attribute does, use the documentation tool in Xcode to look up the object's class and review the descriptions of its properties.

By the Way

Setting Accessibility Attributes

For many years, the "appearance" of an interface meant just how it looks visually. Today, the technology is available for an interface to vocally describe itself to the visually impaired. iOS includes Apple's screen-reader technology: Voiceover. Voiceover combines speech synthesis with a customized interface to aid users in navigating applications.

Using Voiceover, users can touch interface elements and hear a short description of what they do and how they can be used. Although you gain much of this functionality "for free" (the iOS Voiceover software will read button labels, for example), you can provide additional assistance by configuring the accessibility attributes in Interface Builder.

To access the Accessibility settings, you need to open the Identity Inspector by clicking the window icon at the top of the Utility area. You can also choose View, Utilities, Show Identity Inspector or press Option+Command+3. The Accessibility options have their own section within the Identity Inspector, as shown in Figure 5.11.

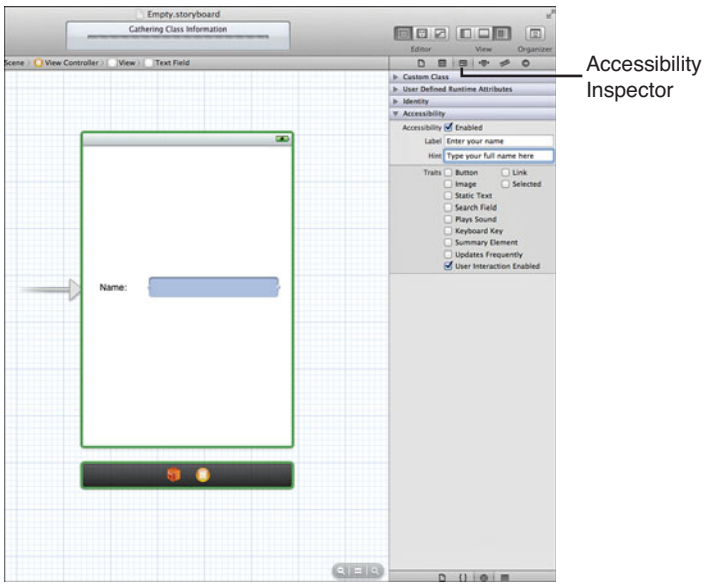

FIGURE 5.11
Use the Accessibility section in the Identity Inspector to configure how Voiceover interacts with your application.

You can configure four sets of attributes within this area:

Accessibility: If enabled, the object is considered accessible. If you create any custom controls that must be seen to be used, this setting should be disabled.

Label: A simple word or two that serves as the label for an item. A text field that collects the user's name might use "your name," for example.

Hint: A short description, if needed, on how to use the control. This is needed only if the label doesn't provide enough information on its own.

Traits: This set of check boxes is used to describe the features of the object—what it does and what its current state is.

Did You Know?

For an application to be available to the largest possible audience, take advantage of accessibility tools whenever possible. Even objects such as the text labels you've used in this lesson should have their traits configured to indicate that they are static text. This helps potential users know that they can't interact with them.

Simulating the Interface

If you've worked with earlier versions of Xcode, you know that you could easily simulate your user interface. Unfortunately, when Apple introduced Storyboards, they removed this capability. *However*, Xcode will now write much of your interface code for you. This means that when you create an interface and connect it to your application classes, you can run the app in the iOS Simulator even though it isn't done. We will follow a development pattern throughout the book that takes advantage of this. Except in a few very unusual instances, you can run your apps at any time to test the interface and any functionality you've added.

Enabling the iOS Accessibility Inspector

If you are building accessible interfaces, you may want to enable the Accessibility Inspector in the iOS Simulator. To do this, start the simulator and click the Home button to return to the home screen. Start the Settings application and navigate to General, Accessibility, and then use the toggle button to turn the Accessibility Inspector on, as shown in Figure 5.12.

The Accessibility Inspector adds an overlay to the simulator workspace that displays the label, hints, and traits that you've configured for your interface elements. Note that navigating the iOS interface is *very* different when operating in accessibility mode.

Using the X button in the upper-left corner of the inspector, you can toggle it on and off. When off, the inspector collapses to a small bar, and the iPhone simulator will behave normally. Clicking the X button again turns it back on. To disable the Accessibility Inspector altogether, just revisit the Accessibility setting in the Settings application.

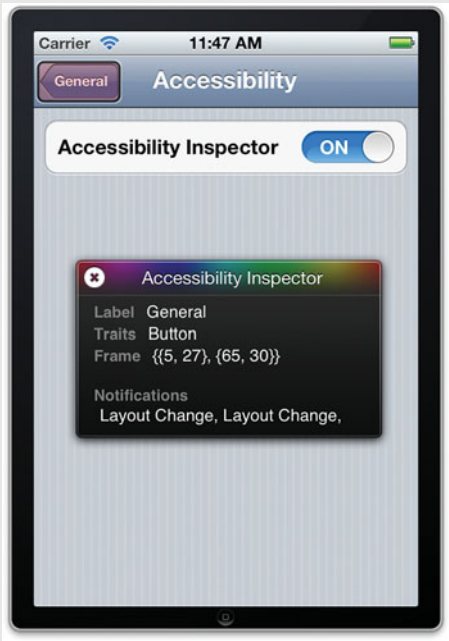

FIGURE 5.12
Toggle the iOS Accessibility Inspector on.

Connecting to Code

You know how to make an interface, but how do you make it *do* something? Throughout this hour, I've been alluding to the idea that connecting an interface to the code you write is just a matter of "connecting the dots." In this last part of the hour, we do just that: take an interface and connect it to the code that makes it into a functional application.

Opening the Project

To get started, we'll use the project Disconnected contained within this hour's Projects folder. Open the folder and double-click the Disconnected.xcodeproj file. This opens the project in Xcode, as shown in Figure 5.13.

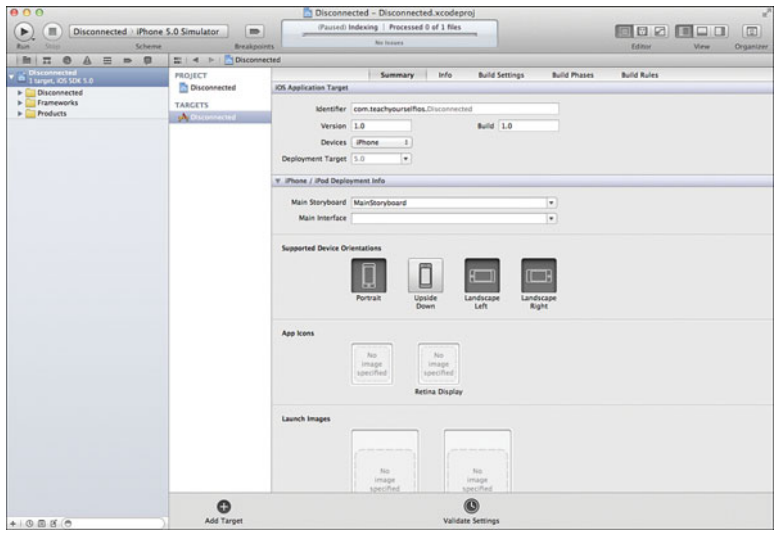

FIGURE 5.13 To begin, open the project in Xcode.

Once the project is loaded, expand the project code group (Disconnected) and click the MainStoryboard.storyboard file. This storyboard file contains the scene and view that this application displays as its interface. Xcode refreshes and displays the scene in the Interface Builder Editor, as shown in Figure 5.14.

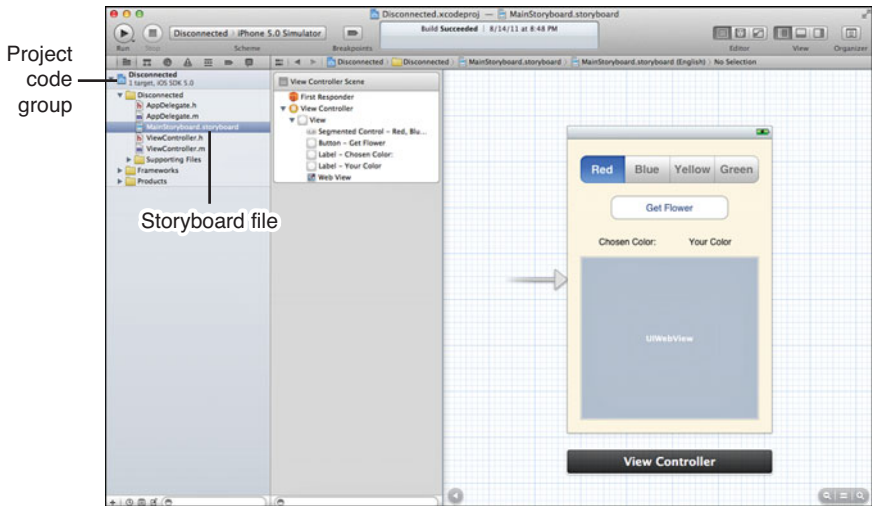

FIGURE 5.14 The Interface Builder Editor displays the scene and corresponding view for the application.

Implementation Overview

The interface contains four interactive elements: a button bar (called a *segmented control*), a push button, an output label, and a web view (an integrated web browser component). Together, these controls interface with application code to enable a user to pick a flower color, touch the Get Flower button, and then display the chosen color in a text label along with a matching flower photo fetched from the website http://www.floraphotographs.com. Figure 5.15 shows the final result.

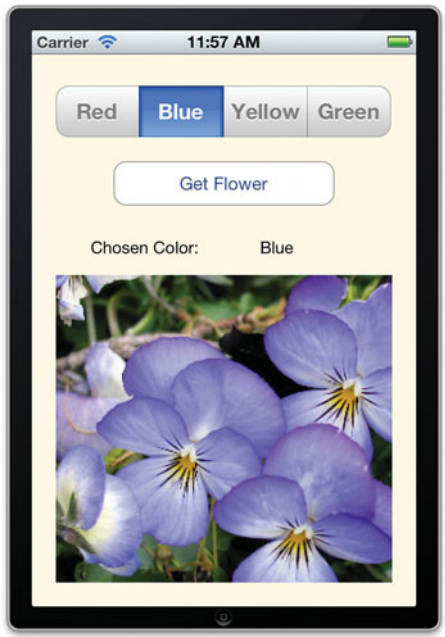

FIGURE 5.15
The finished application will enable a user to choose a color and have a flower image returned that matches that color.

Unfortunately, right now the application does nothing. The interface isn't connected to any application code, so it is hardly more than a pretty picture. To make it work, we'll be creating connections to outlets and actions that have been defined in the application's code.

Outlets and Actions

An *outlet* is nothing more than a variable by which an object can be referenced. For example, if you had created a field in Interface Builder intending that it would be used to collect a user's name, you might want to create an outlet for it in your code called `userName`. Using this outlet and a corresponding property, you could then access or change the contents of the field.

An *action*, on the other hand, is a method within your code that is called when an event takes place. Certain objects, such as buttons and switches, can trigger actions when a user interacts with them through an event, such as touching the screen. If you define actions in your code, Interface Builder can make them available to the onscreen objects.

Joining an element in Interface Builder to an outlet or action creates what is generically termed a *connection*.

For the Disconnected app to function, we need to create connections to these outlets and actions:

- **`ColorChoice`**: An outlet created for the button bar to access the color the user has selected
- **`GetFlower`**: An action that retrieves a flower from the Web, displays it, and updates the label with the chosen color
- **`ChosenColor`**: An outlet for the label that will be updated by `getFlower` to show the name of the chosen color
- **`FlowerView`**: An outlet for the web view that will be updated by `getFlower` to show the image

Let's make the connections now.

Creating Connections to Outlets

To create a connection from an interface item to an outlet, Control-drag from a scene's View Controller icon (in the Document Outline area or the icon bar below the view) to either the visual representation of the object in the view or its icon in the Document Outline area.

Try this with the button bar (segmented control). Pressing Control, click and drag from the View Controller in the Document Outline area to the onscreen image of the bar. A line appears as you drag, enabling you to easily point to the object that you want to use for the connect, as shown in Figure 5.16.

When you release the mouse button, the available connections are shown in a pop-up menu (see Figure 5.17). In this case, you want to pick `colorChoice`.

By the Way

> Interface Builder knows what type of object is allowed to connect to a given outlet, so it displays only the outlets appropriate for the connection you're trying to make.

Repeat this process for the label with the text `Your Color`, connecting it to the `chosenColor` outlet, and the web view, connecting to `flowerView`.

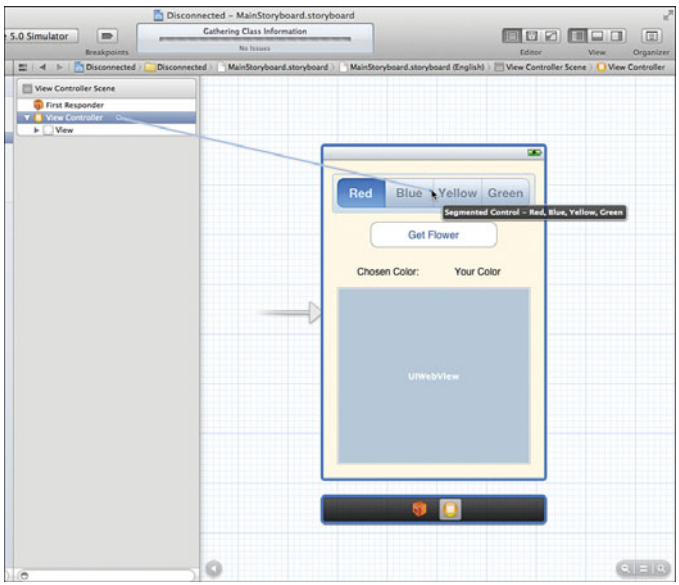

FIGURE 5.16
Control-drag from the View Controller to the button bar.

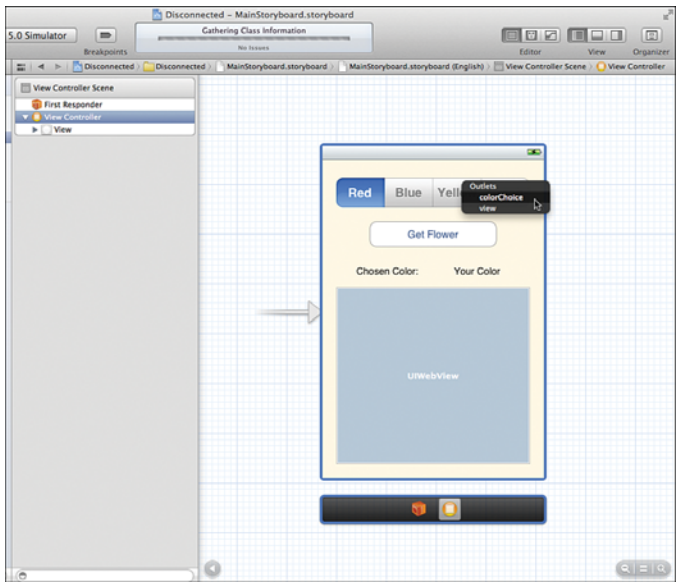

FIGURE 5.17
Choose from the outlets available for the targeted object.

Connecting to Actions

Connecting to actions is a bit different. An object's events trigger actions (methods) in your code. So, the connection direction reverses; you connect from the object invoking an event to the View Controller of its scene. Although it is possible to Control-drag and create a connection in the same manner you did with outlets, this

isn't recommended because you don't get to specify which event triggers it. Do users have to touch the button? Release their fingers from a button?

Actions can be triggered by *many* different events, so you need to make sure that you're picking exactly the right one, instead of leaving it up to Interface Builder. To do this, select the object that will be connecting to the action and open the Connections Inspector by clicking the arrow icon at the top of the Xcode Utility area. You can also show the inspector by choosing View, Utilities, Show Connections Inspector (or by pressing Option+Command+6).

The Connections Inspector, in Figure 5.18, shows a list of the events that the object, in this case a button, supports. Beside each event is an open circle. To connect an event to an action in your code, click and drag from one of these circles to the scene's View Controller icon in the Document Outline area.

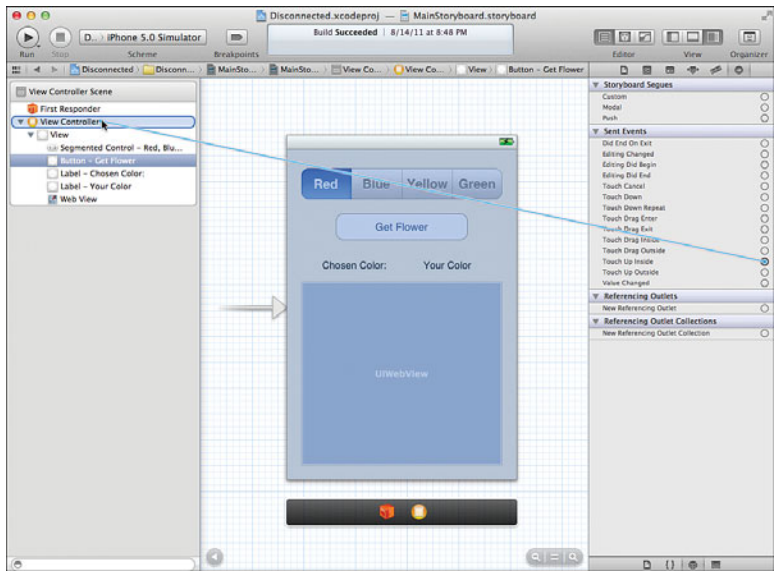

FIGURE 5.18 Use the Connections Inspector to view existing connections and to make new ones.

By the Way

I often refer to creating connections to a *scene's* View Controller or placing interface elements in a *scene's* view. This is because Interface Builder storyboards can contain multiple different scenes, each with its own View Controller and view. In the first few lessons, there is only a single scene, and therefore, a single View Controller. That said, you should still be getting used to the idea of multiple View Controller icons appearing in the Document Outline area and having to correctly choose the one that corresponds to the scene you are editing.

For example, to connect the Get Flower button to the `getFlower` method, select the button, and then open the Connections Inspector (Option+Command+6). Drag from the circle beside the Touch Up Inside event to the scene's View Controller and release, as demonstrated in Figure 5.18. When prompted, choose the `getFlower` action, shown in Figure 5.19.

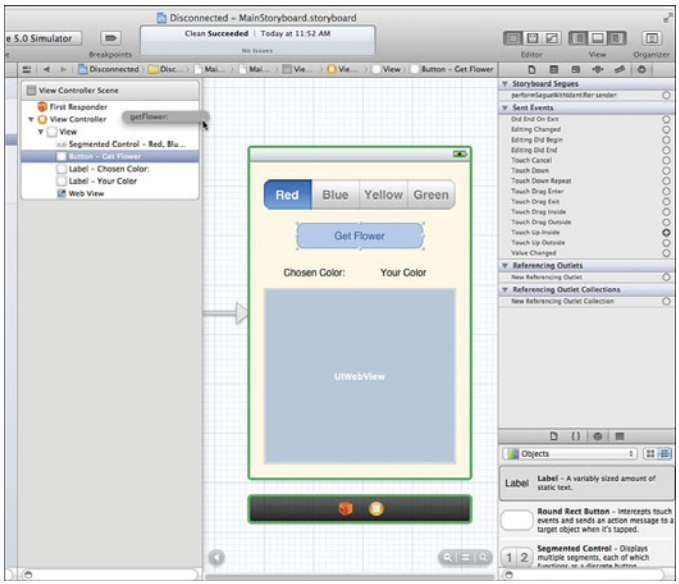

FIGURE 5.19
Choose the action you want the interface element to invoke.

After a connection has been made, the inspector updates to show the event and the action that it calls, demonstrated in Figure 5.20. If you click other already-connected objects, you'll notice that the Connections Inspector shows their connections to outlets and to actions.

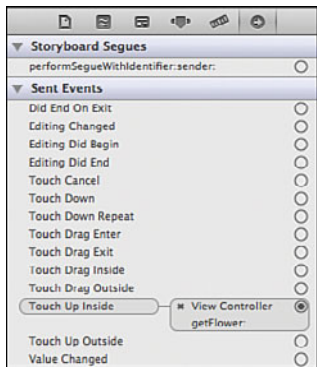

FIGURE 5.20
The Connections Inspector updates to show the actions and outlets that an object references.

Well done! You've just linked an interface to the code that supports it. Click Run on the Xcode toolbar to build and run your application in the iOS Simulator or your personal iDevice.

Connections Without Code

Although most of your connections in Interface Builder will be between objects and outlets and actions you've defined in your code, certain objects implement built-in actions that don't require you to write a single line of code.

The web view, for example, implements actions, including `goForward` and `goBack`. Using these actions, you could add basic navigation functionality to a web view by dragging from a button's Touch Up Inside event directly to the web view object (rather than the view controller). As described previously, you are prompted for the action to connect to, but this time, it isn't an action you had to code yourself.

Editing Connections with the Quick Inspector

One of the errors that I commonly make when connecting my interfaces is creating a connection that I didn't intend. A bit of overzealous dragging, and suddenly your interface is wired up incorrectly and won't work. To review the connections that are in place, you select an object and use the Connections Inspector discussed previously, or you can open the Quick Inspector by right-clicking any object in the Interface Builder editor or Document Outline area. This opens a floating window that contains all the outlets and actions either referenced or received by the object, as shown in Figure 5.21.

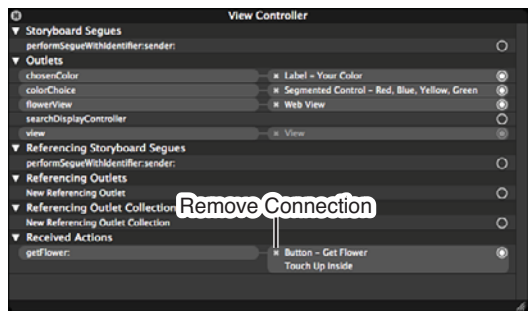

FIGURE 5.21
Right-click to quickly inspect any object connections.

Besides viewing the connections that are in place, you can remove a connection by clicking the X next to a connected object (see Figure 5.21). You can even create new connections using the same "click-and-drag from the circle to an object" approach that you performed with the Connections Inspector. Click the X in the upper-left corner of the window to close the Quick Inspector.

By the Way

Although clicking an object, such as a button, shows you all the connections related to that object, it doesn't show you *everything* you've connected in the Interface Builder Editor. Because almost all the connections you create will go to and from a scene's View Controller, choosing it, then opening the inspector will give you a more complete picture of what connections you've made.

Writing Code with Interface Builder

You just created connections from user interface objects to the corresponding outlets and actions that have already been defined in code. In the next hour's lesson, you write a full application, including defining outlets and actions and connecting them to a storyboard scene. What's interesting about this process, besides it bringing all of the earlier lessons together, is that Interface Builder Editor writes and inserts the necessary Objective-C code to define outlets and actions.

Although it is impossible for Xcode to write your application for you, it does create the instance variables and properties for your app's interface objects, as well as "stubs" of the methods your interface will trigger. All you need to do is drag and drop the Interface Builder objects into your source code files. Using this feature is completely optional, but it does help save time and avoid syntax errors.

Did You Know?

A method *stub* (or *skeleton*) is nothing more than a method that has been declared but executes no instructions. You can add stubs to your code where you know what you'll be writing in the future but aren't yet ready to commit it to code. This is useful in the initial design stages of an application because it helps you keep track of the work you have left to do.

Stub methods are also helpful if you have code that needs to use a method that you haven't written. By inserting and referencing stubs for your unwritten methods, your application will compile and run—enabling the code that *is* complete to be tested at any stage of the development process.

Object Identity

As we finish up our introduction to Interface Builder, I'd be remiss if I didn't introduce one more feature: the Identity Inspector. You've already accessed this tool to view the accessibility attributes for interface objects, but there is another reason why we'll need to use the inspector in the future: setting class identities and labels.

As you drag objects into the interface, you're creating instances of classes that already exist (buttons, labels, and so on). Throughout this book, however, we build custom subclasses that we also need to be able to reference with Interface Builder's objects. In these cases, we need to help Interface Builder by identifying the subclass it should use.

For example, suppose we created a subclass of the standard button class (`UIButton`) that we named `ourFancyButtonClass`. We might drag a button into a scene to represent our fancy button, but when the storyboard file loads, it would just create the same old `UIButton`.

To fix the problem, we select the button we've added to the view, open the Identity Inspector by clicking the window icon at the top of the Xcode Utility area or by choosing View, Utilities, Show Identity Inspector (Option+Command+3), and then use the drop-down menu/field to enter the class that we really want instantiated at runtime (see Figure 5.22).

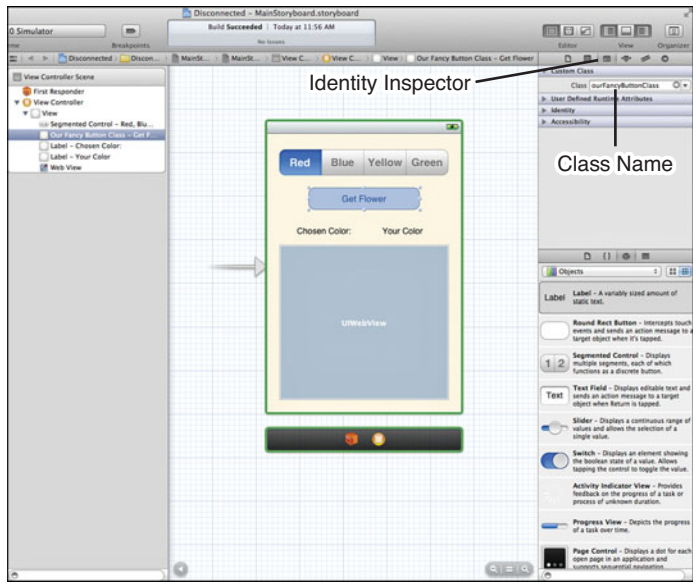

FIGURE 5.22
If you're using a custom class, you'll need to manually set the identity of your objects in the Identity Inspector.

This is something we'll cover on an as-needed basis, so if it seems confusing, don't worry. We come back to it later in the book.

Further Exploration

The Interface Builder Editor gives you the opportunity to experiment with many of the different GUI objects you've seen in iOS applications and read about in the previous hours. In the next hour, the Xcode code editor is used in conjunction with the Xcode Interface Builder for your first full project, developed from start to finish.

To learn even more about what you can do with Interface Builder, I suggest reading through the following three Apple publications:

Interface Builder Help: Accessed by right-clicking the background in the Interface Builder Editor, the IB help is more than a simple help document. Apple's Interface Builder Help walks you through the intricacies of IB using video tutorials and covers some advanced topics that will be important as your development experience increases.

iOS Human Interface Guidelines: The Apple iOS HIG document provides a clear set of rules for building usable interfaces on the iOS device family. This document describes when you should use controls and how they should be displayed, helping you create more polished, professional-quality applications.

Accessibility Programming Guide for iOS: If you're serious about creating accessible apps, this is a mandatory read. The Accessibility Programming Guide describes the accessibility features mentioned in this hour's lesson as well as ways to improve accessibility programmatically and methods of testing accessibility beyond the tips given in this hour.

As a general note, from here on, you do quite a bit of coding in each lesson. So now is a great time to review the previous hours if you have any questions.

Summary

In this hour, you explored the Xcode Interface Builder Editor and the tools it provides for building rich graphical interfaces for your iOS applications. You learned how to navigate IB storyboards and access the GUI elements from the Object Library. Using the various inspector tools within Interface Builder, you customized the look and feel of the onscreen controls and how they can be made accessible to the visually impaired.

More than just a pretty picture, an IB-created interface uses simple outlets and actions to connect to functionality in your code. You used Interface Builder's connection tools to turn a nonfunctioning interface into a complete application. By maintaining a separation between the code you write and what is displayed to the user, you can revise your interface to look however you want, without breaking your application. In Hour 6, you examine how to create outlets and actions from scratch in Xcode (and thus gain a full toolset to get started developing).

Q&A

Q. ***Why do I keep seeing things referred to as NIB/XIB files?***

A. The origins of Interface Builder trace back to the NeXT Computer, which made use of NIB files to store individual views. These files, in fact, still bore the same name when Mac OS X was released. In recent years, however, Apple renamed the files to have the .xib extension, which has subsequently been replaced by storyboards and scenes. Unfortunately, Apple's documentation hasn't quite caught up yet and may reference XIB or NIB files. If you encounter these documentation barnacles, just substitute "storyboard scene" for XIB or NIB in your head.

Q. ***Some of the objects in the Interface Builder Object Library can't be added to my view. What gives?***

A. Not all the items in the Object Library are interface objects. Some represent objects that provide functionality to your application. These can be added to the scene in the Document Outline area or on the icon bar located below a scene's layout in the IB editor.

Q. ***I've seen controls in applications that aren't available here. Where are they?***

A. Keep in mind that the iOS objects are heavily customizable and frequently used as a starting point for developers to make their own UI classes or subclasses. The end result can vary tremendously from the stock UI appearance.

Workshop

Quiz

1. Simulating a scene using IB's Simulate Document feature also compiles the project's code in Xcode. True or false?
2. What tool can you use within the iOS Simulator to help review accessibility of objects in your apps?
3. What two connection types can be made in the Xcode Interface Builder?

Answers

1. False. Simulating a scene does not use the project code at all. As a result, the interface will not perform any actions that rely on underlying code.
2. The Accessibility Inspector makes it possible to view the accessibility attributes configured within Interface Builder.
3. Connections to outlets and actions can be created in Interface Builder. A connection to an outlet provides a means of referencing and working with a UI element in code. A connection to an action defines a UI event, such as a button press, that will execute the action's method.

Activities

1. Practice using the interface layout tools on the Empty.storyboard file. Add each available interface object to your view, and then review the Attributes Inspector for that object. If an attribute doesn't make sense, remember that you can review documentation for the class to identify the role of each of its properties.
2. Revise the Disconnected project with an accessible interface. Review the finished design using the Accessibility Inspector in the iOS Simulator.

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

Y–Z