HOUR 8

Handling Images, Animation, and Sliders

What You'll Learn in This Hour:

- The use of sliders for user input
- Configuring and manipulating the slider input range
- How to add image views to your projects
- Ways of creating and controlling simple animations

The text input and output that you learned about in the preceding hour is certainly important, but the iPhone is known for its attractive graphics and "touchable" UI. This hour expands our interface toolkit to include images, animation, and the very touchable slider control.

We'll be implementing an application to combine these new features along with simple logic to manipulate input data in a unique way. These new capabilities will help you build more interesting and interactive applications—and, of course, there's more to come in the next hour!

User Input and Output

Although application logic is always the most important part of an application, the way the interface works plays a big part in how well it will be received. For Apple and the iPhone, providing a fun, smooth, and beautiful user experience has been key to its success; it's up to you to bring this experience into your own development. The iPhone SDK's interface options give you the tools to express your application's functionality in fun and unique ways.

This hour introduces two very visual interface features: sliders for input and image views for output.

Sliders

The first new interface component that we'll be using this hour is a slider (`UISlider`). Sliders are a convenient touch control that is used to visually set a point within a range of values. Huh? What?

Suppose that you want your user to be able to speed something up or slow it down. Asking users to input timing values is unreasonable. Instead, you can present a slider, as seen in Figure 8.1, where they can touch and drag an indicator back and forth on a line. Behind the scenes, a `value` property is being set that your application can access and use to set the speed. No need for users to understand the behind-the-scene details or do anything more than drag with their fingers.

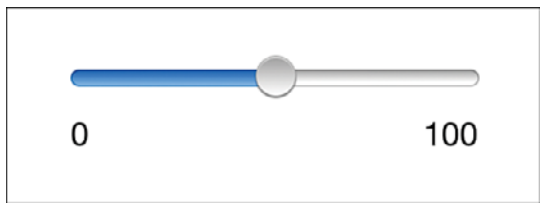

FIGURE 8.1
Use a slider to collect a value from a range of numbers without requiring users to type.

Sliders, like buttons, can react to events or can be read passively like a text field. If you want the user's changes to a slider to immediately have an effect on your application, you must have it trigger an action.

Image Views

Image views (`UIImageView`) do precisely what you'd think: They display images! They can be added to your application views and used to present information to the user. An instance of `UIImageView` can even be used to create a simple frame-based animation with controls for starting, stopping, and even setting the speed at which the animation is shown.

With iOS 4, your image views can even take advantage of the high-resolution display of the iPhone 4 (and any other upcoming iOS high-resolution devices). Even better, you need no special coding! Instead of checking for a specific device, you can just add multiple images to your project, and the image view will load the right one at the right time. We won't go through all the steps to make this happen each time we use an image in this book, but I do describe how you can add this capability to your projects later in this hour's lesson.

Creating and Managing Image Animations and Sliders

There's something about interface components that *move* that make users take notice. They're visually interesting, attract and keep attention, and, on the iPhone's touch screen, are fun to play with. In this hour's project, we take advantage of both of our new UI elements (and some old friends) to create a user-controlled animation.

Implementation Overview

As mentioned earlier, image views can be used to display image file resources and show simple animations, whereas sliders provide a visual way to choose a value from a range. We'll combine these in an application we're calling ImageHop.

In ImageHop, we'll be creating a looping animation using a series of images and an image view instance (`UIImageView`). We'll allow the user to set the speed of the animation using a slider (`UISlider`). What will we be using as an animation? A hopping bunny. What will the user control? Hops per second, of course! The "hops" value set by the slider will be displayed in a label (`UILabel`). The user will also be able to stop or start the animation using a button (`UIButton`).

Figure 8.2 shows the completed application in use.

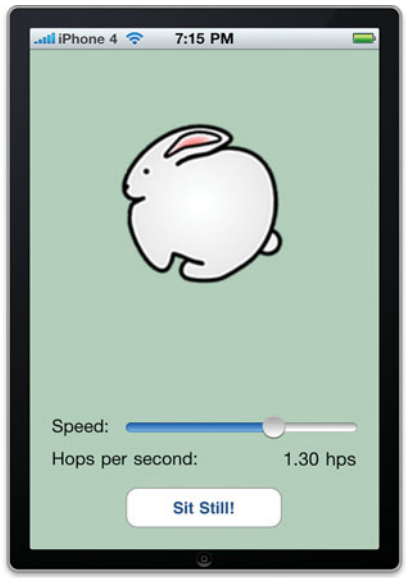

FIGURE 8.2
ImageHop uses an image view and a slider to create and control a simple animation.

We should discuss two pieces of this project before getting too far into the implementation:

- First, image view animations are created using a series of images. I've provided a 20-frame animation with this project, but you're welcome to use your own images if you prefer.
- Second, although sliders enable users to visually enter a value from a range, there isn't much control over how that is accomplished. For example, the minimum value must be smaller than the maximum, and you can't control which dragging direction of the slider increases or decreases the result value. These limitations aren't show-stoppers; they just mean that there may be a bit of math (or experimentation) involved to get the behavior you want.

Setting Up the Project

Begin this project in the same way as the last. Launch Xcode (Developer/Applications), and then choose File, New Project.

Select the iPhone OS Application project type, and then find and select the View-Based Application option in the Template list on the right. Click Choose to continue, enter the project name **ImageHop**, and save the new project.

Adding the Animation Resources

This project makes use of 20 frames of animation stored as PNG files. The frames are included in the Images folder within the ImageHop project folder.

Because we know up front that we'll need these images, drag them into the Xcode project's Resources folder, being sure to choose the option to copy the resources if needed.

Preparing the Outlets and Actions

In this application, we need to provide outlets and actions for several objects.

For outlets, first we need the image view (`UIImageView`), which will contain the animation and be referenced through the variable `imageView`. The slider control (`UISlider`) will set the speed and will be connected via `animationSpeed`, while the speed value itself will be output in a label named `hopsPerSecond` (`UILabel`). A button (`UIButton`) will toggle the animation on and off and will be connected to an outlet `toggleButton`.

By the Way

Why do we need an outlet for the button? Shouldn't it just be triggering an action to toggle the animation? Yes, the button could be implemented without an outlet, but by including an outlet for it, we have a convenient way of setting the button's title in the code. We can use this to change the button to read "Stop" when the image is animating or "Start" when the animation has stopped.

For actions, we need only two: `setSpeed` will be the method called when the slider value has changed and the animation speed needs to be reset, and `toggleAnimation` will be used to start and stop the animation sequence.

Go ahead and define these outlets and actions as outlets and actions within ImageHopViewController.h. You'll also want to declare the four outlet variables as properties so that we can easily access them in the view controller code. Listing 8.1 shows the resulting header file.

LISTING 8.1

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

@interface ImageHopViewController : UIViewController {
    IBOutlet UIImageView *imageView;
    IBOutlet UIButton *toggleButton;
    IBOutlet UISlider *animationSpeed;
    IBOutlet UILabel *hopsPerSecond;
}

@property (retain,nonatomic) UIImageView *imageView;
@property (retain,nonatomic) UIButton *toggleButton;
@property (retain,nonatomic) UISlider *animationSpeed;
@property (retain,nonatomic) UILabel *hopsPerSecond;

-(IBAction)toggleAnimation:(id)sender;
-(IBAction)setSpeed:(id)sender;

@end
```

For all the properties you've defined in the header file, add an `@synthesize` directive in the ImageHopViewController.m implementation file. Your additions should fall after the `@implementation` line and look like this:

```
@synthesize toggleButton;
@synthesize imageView;
@synthesize animationSpeed;
@synthesize hopsPerSecond;
```

Make sure that both the `ImageHopViewController` header and implementation files have been saved, and then launch Interface Builder by double-clicking the ImageHopViewController.xib file within the project's Resources folder.

After it has loaded, switch to the Document window (Window, Document), and double-click the view icon to open it and begin editing.

Adding an Image View

In this exercise, our view creation will begin with the most important object of the project: the image view (`UIImageView`). Open the Interface Builder Objects Library and drag an image view into the view window.

Because the view is has no images assigned, it will be represented by a light-gray rectangle. Use the resize handles on the rectangle to size it to fit in the upper two-thirds of the interface (see Figure 8.3).

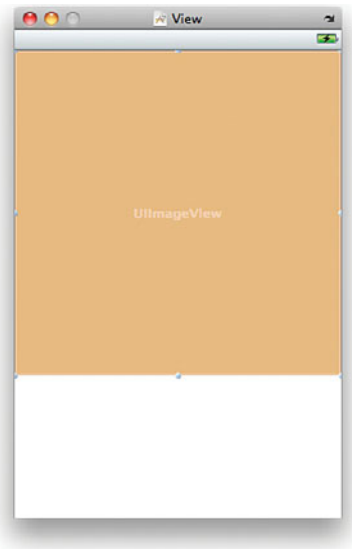

FIGURE 8.3
Set the image view to fill the upper two-thirds of the iPhone interface.

Setting the Default Image

There are very few attributes for configuring the functionality of an image view. In fact, there is only one: the image that is going to be displayed. Select the image view and press Command+1 to open the Attributes Inspector (see Figure 8.4).

Using the Image drop-down menu, choose one of the image resources available. This will be the image that is shown before the animation runs, so using the first frame (frame-1.png) is a good choice.

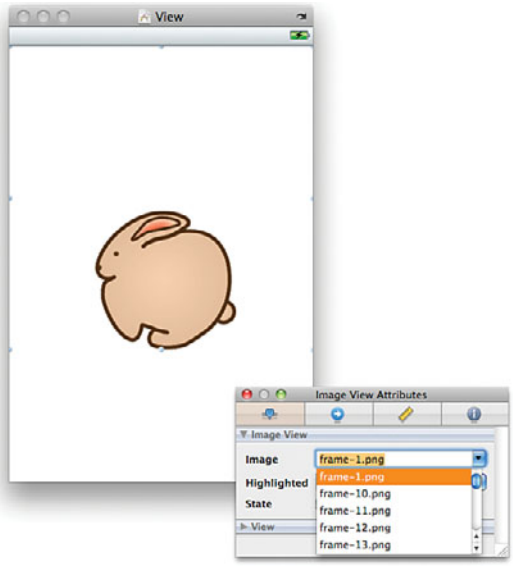

FIGURE 8.4
Set the image that will be shown in the view.

By the Way

What about the animation? Isn't this just a frame? Yes, if we don't do anything else, the image view will show a single static image. To display an animation, we need to create an array with all the frames and supply it programmatically to the image view object. We do this in a few minutes, so just hang in there!

The image view will update in Interface Builder to show the image resource that you've chosen.

You Said You'd Tell Us About Loading Hi-Res Images for the iPhone 4. How Do We Do It?

That's the best part! There's really nothing to do that you don't already know. To accommodate the higher scaling factor of the iPhone 4, you just create image resources that are two times the horizontal and vertical resolution, and then name them with the same filename as your original low-res images, but with the suffix @2x (for example, Image.png becomes Image@2x.png). Finally, add them to your project resources like any other resource.

Within your projects, just reference the low-res image, and the hi-res image is loaded automatically on the correct devices, as needed!

Connecting to the Outlet

To display an animation, we need to access the object from the ImageHop view controller. Let's connect the image view to the `imageView` outlet that we created earlier.

Within the Document window, Control-drag from the File's Owner icon to the image view icon in the Document window or to the graphical representation in the view window. When prompted for the outlet, choose `imageView`, as shown in Figure 8.5.

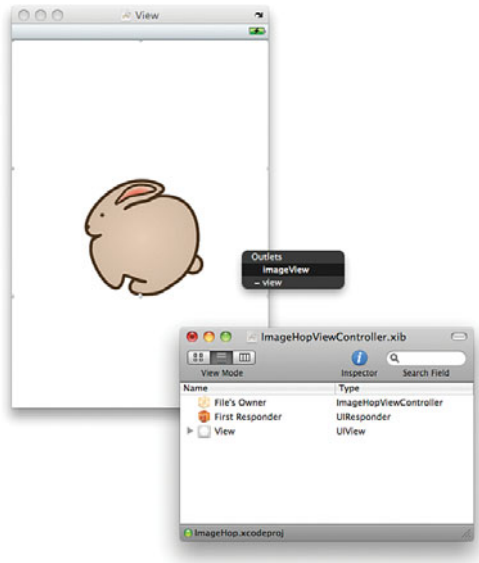

FIGURE 8.5
Connect the image view to an outlet so that it can be easily accessed from code.

Now that the image view has been added, let's look at the code we need to add to change from a static image to an animation.

Animating the Image View

To truly customize an image view, we need to write some code. Animating images requires us to build an array of image objects (`UIImage`) and pass them to the image view. Where should we do this? As with the last project, the `ViewDidLoad` method of our view controller provides a convenient location for doing additional setup for the view, so that's what we'll use.

Switch back into Xcode, and open the view controller implementation file, ImageHopViewController.m. Find the `ViewDidLoad` method and uncomment it, and then add the following code to the method. Note that we've removed lines 7–20 to save space (they follow the same pattern as lines 4–6 and 21–23), as shown in Listing 8.2.

LISTING 8.2

```
 1: - (void)viewDidLoad {
 2:     NSArray *hopAnimation;
 3:     hopAnimation=[[NSArray alloc] initWithObjects:
 4:                   [UIImage imageNamed:@"frame-1.png"],
 5:                   [UIImage imageNamed:@"frame-2.png"],
 6:                   [UIImage imageNamed:@"frame-3.png"],
…
21:                   [UIImage imageNamed:@"frame-18.png"],
22:                   [UIImage imageNamed:@"frame-19.png"],
23:                   [UIImage imageNamed:@"frame-20.png"],
24:                   nil
25:                  ];
26:     imageView.animationImages=hopAnimation;
27:     imageView.animationDuration=1;
28:     [hopAnimation release];
29:     [super viewDidLoad];
30: }
```

To configure the image view for animation, first an array (`NSArray`) variable is declared (line 2) called `hopAnimation`. Next, in line 3, the array is allocated and initialized via the `NSArray` instance method `initWithObjects`. This method takes a comma-separated list of objects, ending with `nil`, and returns an array.

The image objects (`UIImage`) are initialized and added to the array in lines 4–24. Remember that you'll need to fill in lines 7–20 on your own; otherwise, several frames will be missing from the animation!

Once an array is populated with image objects, it can be used to set up the animation of an image view. To do this, set the `animationImages` property of the image view (`imageView`) to the array. Line 6 accomplishes this for our example project.

Another image view property that we'll want to set right away is the `animationDuration`. This is the number of seconds it takes for a single cycle of the animation to be played. If the duration is *not* set, the playback rate will be 30 frames per second. To start, our animation will be set to play all the frames in 1 second, so line 27 sets the `imageView.animationDuration` to `1`.

Finally, in line 28, we're finished with the `hopAnimation` array, so it can be released.

Starting and Stopping the Animation

A little later in this tutorial, we'll be adding controls to change the animation speed and to start/stop the animation loop. You've just learned how the `animationDuration` property can change the animation speed, but we'll need three more properties/methods to accomplish everything we want:

> `isAnimating`: This property returns `true` if the image view is currently animating its contents.

`startAnimating`: Starts the animation.

`stopAnimating`: Stops the animation if it is running.

If you run the application now, it will work, but only a static image will display. The image view does not start animating until the `startAnimating` method is called. We'll take care of that when implementing the view controller logic.

Adding a Slider

The next piece that our interface needs is the slider that will control the speed. Return to Interface Builder and the view, and then navigate to the Objects Library and drag the slider (`UISlider`) into the view, just under the image view. Using the resize handles on the slider, click and drag to size it to about two-thirds of the image view width and align it with the right side of the image view. This leaves just enough room for a label to the left of the slider.

Because a slider has no visual indication of its purpose, it's a good idea to always label sliders so that your users will understand what they do. Drag a label object (`UILabel`) from the Library into your view. Double-click the text and set it to read **Speed:**. Position it so that it is aligned with the slider, as shown in Figure 8.6.

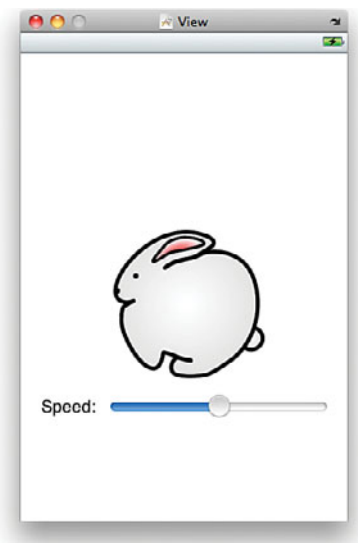

FIGURE 8.6
Add the slider and a corresponding label to the view.

Setting the Slider Range Attributes

Sliders make their current settings available through a `value` property that we'll be accessing in the view controller. To change the range of values that can be returned,

we need to edit the slider attributes. Click to select the slider in the view, and then open the Attributes Inspector (Command+1), as shown in Figure 8.7.

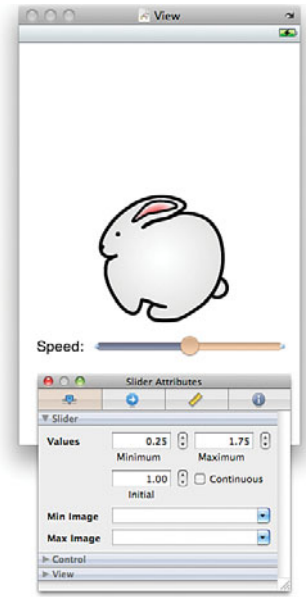

FIGURE 8.7
Edit the slider's attributes to control the range of values it returns.

The Minimum, Maximum, and Initial fields should be changed to contain the smallest, largest, and starting values for the slider. For this project, use .25, 1.75, and 1.0, respectively.

Where Did These Min, Max, and Initial Values Come From?

This is a great question, and one that doesn't have a clearly defined answer. In this application, the slider represents the speed of the animation, which, as we've discussed, is set through the `animationDuration` property of the image view as the number of seconds it takes to show a full cycle of an animation. Unfortunately, this means the *faster* animations would use smaller numbers and *slower* animations use larger numbers, which is the exact opposite of traditional user interfaces where "slow" is on the left and "fast" is on the right. Because of this, we need to reverse the scale. In other words, we want the big number (1.75) to appear when the slider is on the left side and the small number (.25) on the right.

To reverse the scale, we take the combined total of the minimum and maximum (1.75 + 0.25), and subtract the value returned by the slider from that total. For example, when the slider returns 1.75 at the top of the scale, we'll calculate a duration of 2 – 1.75, or 0.25. At the bottom of the scale, the calculation will be 2 – 0.25, or 1.75.

Our initial value will be 1.0, which falls directly in the middle of the scale.

Make sure the Continuous check box isn't checked. This option, when enabled, will have the control to generate a series of events as the user drags back and forth on the slider. When it isn't enabled, events are generated only when the user lifts his or her finger from the screen. For our application, this makes the most sense and is certainly the least resource-intensive option.

The slider can also be configured with images at the minimum and maximum sliders of the control. Use the Min Image and Max Image drop-downs to select a project image resource if you'd like to use this feature. (We're not using it in this project.)

Connecting to the Outlet

For convenient access to the slider, we created an outlet, `animationSpeed`, that we'll be using in the view controller. To connect the slider to the outlet, Control-drag from the File's Owner icon to the slider object in the view or the slider icon in the Document window. When prompted, choose the `animationSpeed` outlet.

By the Way

In case you're wondering, it's certainly possible to implement this application without an outlet for the slider. When the slider triggers an action, we could use the sender variable to reference the slider `value` property. That said, this approach will allow us to access the slider properties anywhere in the view controller, not just when the slider triggers an action.

Connecting to the Action

When a user drags the slider and releases his finger, the application should trigger the action method `setSpeed`. Create this connection by selecting the slider and then opening the Connections Inspector (Command+2).

Drag from the circle beside Value Changed to the File's Owner icon in the Document window. When prompted, choose to connect to the `setSpeed` action. Once complete, Connections Inspector should reflect this change and show both the `setSpeed` and `animationSpeed` connections, as demonstrated in Figure 8.8.

That completes the major parts of the UI, but there's still some cleanup work to do.

Finishing the Interface

The remaining components of the ImageHop application are interface features that you've used before, so we've saved them for last. We'll finish things up by adding a button to start and stop the animation, along with a readout of the speed of the animated rabbit in "hops per second."

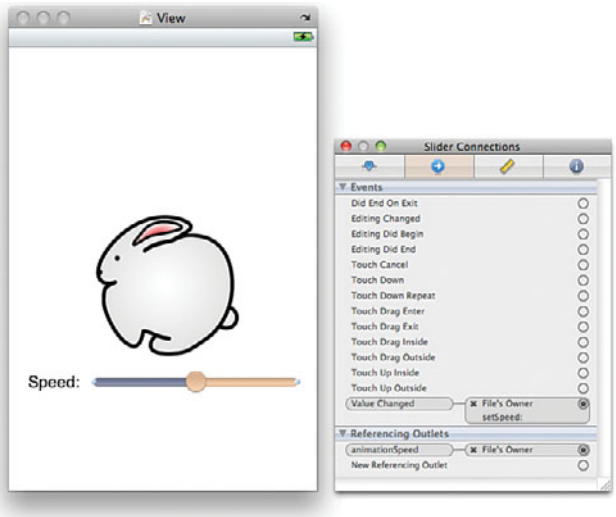

FIGURE 8.8
When the user drags and releases the slider, the `setSpeed` method is called.

Adding Labels

Start by dragging two labels (`UILabel`) to the view. The first label should be set to read **hops per second:** and be located below the slider. Add the second label, which will be used as output of the actual speed value, to the right of the first label.

Change the output label to read **1.00 hps** (the speed that the animation will be starting out at). Using the Attributes Inspector (Command+1), set the text of the label to align right; this will keep the text from jumping around as the user changes the speed.

Finally, Control-drag from the File's Owner icon to the output label, and choose the hopsPerSecond outlet, as shown in Figure 8.9.

Adding the Hop Button

The last part of the ImageHop interface is the button (`UIButton`) that starts and stops the animation. Drag a new button from the Objects Library to the view, positioning it at the bottom center of the UI. Double-click the button to edit the title, and set it to **Hop!**

Like the slider, the hop button needs to be connected to an outlet (`toggleButton`) and an action (`toggleAnimation`). Control-drag from File's Owner icon in the Document window to the button and choose the toggleButton outlet when prompted.

Next, select the button and open the Connections Inspector (Command+2). Within the inspector, click and drag from the circle beside the Touch Up Inside event to the File's Owner icon in the Document window. Connect to the `toggleAnimation` action. Figure 8.10 shows the completed interface and button connections.

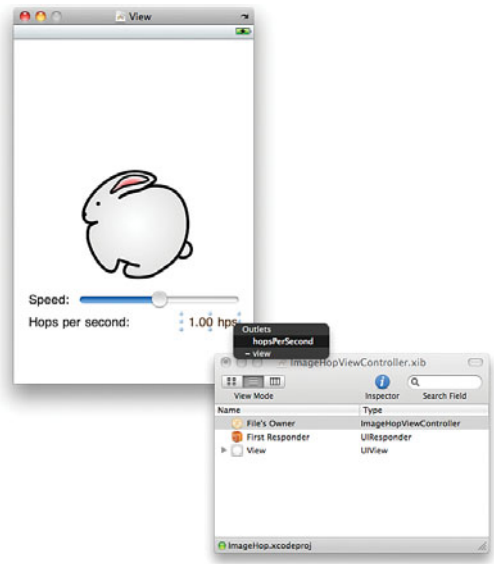

FIGURE 8.9
Connect the label that will be used to display the speed.

The application interface is finished. In the next section, we complete the application by writing the code for starting and stopping the animation and setting the speed.

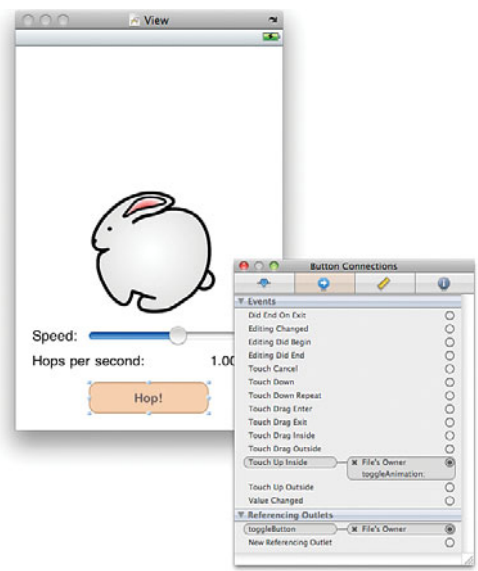

FIGURE 8.10
Connect the button to its outlet and action.

Implementing the View Controller Logic

The `ImageHopViewController` still needs a bit of work before we can call ImageHop done and finally view the animation. Two actions, `toggleAnimation` and `setSpeed`, need to be written. These methods will handle the user's interaction with the ImageHop application through the button and slider, respectively.

Starting and Stopping the Animation

When the user touches the Hop! button, the `toggleAnimation` method is called. This method should use the `isAnimating` property of the image view (`imageView`) to check to see whether an animation is running. If it isn't, the animation should start; otherwise, it should stop. To make sure the user interface makes sense, the button itself (`toggleButton`) should be altered to show the title Sit Still! if the animation is running and Hop! when it isn't.

Add the code in Listing 8.3 to the `ImageHopViewController` implementation file after the `@synthesize` directives.

Listing 8.3

```
1: -(IBAction) toggleAnimation:(id)sender {
2:     if (imageView.isAnimating) {
3:         [imageView stopAnimating];
4:         [toggleButton setTitle:@"Hop!" forState:UIControlStateNormal];
5:     } else {
6:         [imageView startAnimating];
7:         [toggleButton setTitle:@"Sit Still!" forState:UIControlStateNormal];
8:     }
9: }
```

Lines 2 and 5 provide the two different conditions that we need to work with. Lines 3 and 4 are executed if the animation is running, while lines 6 and 7 are executed if it isn't. In line 3 and line 6, the `stopAnimating` and `startAnimating` methods are called for the image view to start and stop the animation, respectively.

Lines 4 and 5 use the `UIButton` instance method `setTitle:forState` to set the button title to the string `"Hop!"` or `"Sit Still!"`. These titles are set for the button state of `UIControlStateNormal`. As you learned earlier this hour, the "normal" state for a button is its default state, prior to any user event taking place.

Setting the Animation Speed

The slider triggers the `setSpeed` action after the user adjusts the slider control. This action must translate into several changes in the actual application: First, the speed of the animation (`animationDuration`) should change. Second, the animation should be started if it isn't already running. Third, the button (`toggleButton`) title

should be updated to show the animation is running. And finally, the speed should be displayed in the `hopsPerSecond` label.

Add the code in Listing 8.4 to the view controller, and then let's review how it works.

LISTING 8.4

```
 1: -(IBAction) setSpeed:(id)sender {
 2:     NSString *hopRateString;
 3:     imageView.animationDuration=2-animationSpeed.value;
 4:     [imageView startAnimating];
 5:     [toggleButton setTitle:@"Sit Still!"
 6:                   forState:UIControlStateNormal];
 7:     hopRateString=[[NSString alloc]
 8:                    initWithFormat:@"%1.2f hps",1/(2-animationSpeed.value)];
 9:     hopsPerSecond.text=hopRateString;
10:     [hopRateString release];
11: }
```

Because we'll need to format a string to display the speed, we kick things off by declaring an `NSString` reference, `hopRateString`, in line 2. In line 3, the image view's (`imageView`) `animationDuration` property is set to 2 minus the value of the slider (`animationSpeed.value`). This, if you recall, is necessary to reverse the scale so that faster is on the right and slower is on the left.

Line 4 uses the `startAnimating` method to start the animation running. Note that it is safe to use this method if the animation is already started, so we don't really need to check the state of the image view. Lines 5 and 6 set the button title to the string `"Sit Still!"` to reflect the animated state.

Lines 7 and 8 allocate and initialize the `hopRateString` instance that we declared in line 2. The string is initialized with a format of `"1.2f"`, based on the calculation of 1 / (2 – `animationSpeed.value`).

Let's break that down a bit further: Remember that the speed of the animation is measured in seconds. The fastest speed we can set is 0.25 (a quarter of a second), meaning that the animation plays 4 times in 1 second (or "4 hops per second"). To calculate this in the application, we simply divide 1 by the chosen animation duration, or 1 / (2 – `animationSpeed.value`). Because this doesn't necessarily return a whole number, we use the `initWithFormat` method to create a string that holds a nicely formatted version of the result. The `initWithFormat` parameter string `"1.2f hps"` is shorthand for saying the number being formatted as a string is a floating-point value (`f`), and that there should always be one digit on the left of the decimal and two digits on the right (`1.2`). The `hps` portion of the format is just the "hops per second" unit that we want to append to the end of the string. For example, if the equation returns a value of .5 (half a hop a second), the string stored in `hopRateString` is set to `"0.50 hps"`.

In line 9, the output label (`UILabel`) in the interface is set to the `hopRateString`. Once finished with the string, line 10 releases it, freeing up the memory it was using.

By the Way

Don't worry if the math here is a bit befuddling. This is not critical to understanding Cocoa or iOS development, it's just an annoying manipulation we needed to perform to get the values the way we want them. I strongly urge you to play with the slider values and calculations as much as you'd like so that you can get a better sense of what is happening here and what steps you might need to take to make the best use of slider ranges in your own applications.

Releasing the Objects

Our development efforts have resulted in four objects that should be released when we're finished: `toggleButton`, `imageView`, `hopsPerSecond`, and `animationSpeed`. Edit the `dealloc` method to release these now:

```
- (void)dealloc {
    [toggleButton release];
    [imageView release];
    [hopsPerSecond release];
    [animationSpeed release];
    [super dealloc];
}
```

Well done! You've just completed the app!

Building the Application

To try your hand at controlling an out-of-control bunny rabbit, click Build and Run in Xcode. After a few seconds, the finished ImageHop application will start, as shown in Figure 8.11.

By the Way

In my version of the tutorial app, I've set the background to a light-green color. Feel free to explore different layouts of the interface within what you've built. You may be surprised how much variation is possible without writing a single additional line of code!

Although ImageHop isn't an application that you're likely to keep on your phone (for long), it did provide you with new tools for your iPhone application toolkit. The `UIImageView` class can easily add dynamic images to your programs, while `UISlider` offers a uniquely touchable input solution.

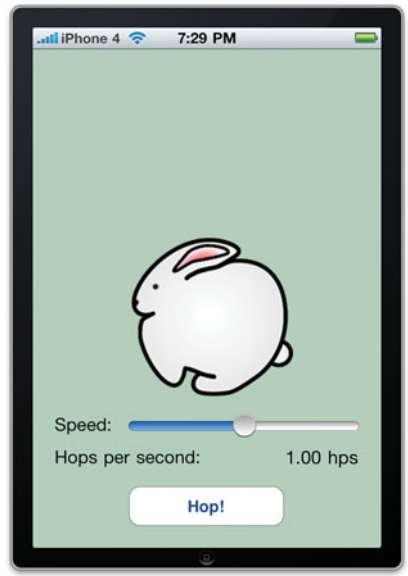

FIGURE 8.11
Bouncing bunnies! What more could we ask for?

Further Exploration

Although many hours in this book focus on adding features to the user interface, it is important to start thinking about the application logic that will bring your user interface to life. As we experienced with our sample application, sometimes creativity is required to make things work the way we want.

Review the properties and methods for `UISlider` class and consider how you might use a slider in your own apps. Can you think of any situations where the slider values couldn't be used directly in your software? How might you apply application logic to map slider values to usable input? Programming is very much about problem solving—you'll rarely write something that doesn't have at least a few "gotchas" that need solved.

In addition to `UISlider`, you may want to review the documentation for `UIImage`. Although we focused on `UIImageView` for displaying our image animation, the images themselves were objects of type `UIImage`. Image objects will come in handy for future interfaces that integrate graphics into the user controls themselves.

Finally, for a complete picture of how your applications will almost automatically take advantage of the higher-resolution display of the iPhone 4, be sure to read the section "Supporting High-Resolution Screens" within the iPhone Application Programming Guide.

Apple Tutorials

`UIImageView`, `UIImage`, `UISlider` - `UICatalog` (accessible via the Xcode documentation). Once again, this project is a great place for exploring any and everything (including images, image views, and sliders) related to the iPhone interface.

Summary

Users of highly visual devices demand highly visual interfaces. In this hour's lesson, you learned about the use of two visual elements that you can begin adding to your applications: image views and sliders. Image views provide a quick means of displaying images that you've added to your project—even using a sequence of images to create animation. Sliders can be used to collect user input from a continuous range of values. These new input/output methods start our exploration of iPhone interfaces that go beyond simple text and buttons.

The information you learned in this hour, although not complex, will help pave the way for mega-rich, touch-centric user interfaces.

Q&A

Q. ***Is the UIImageView the only means of displaying animated movies?***

A. No. The iPhone SDK includes a wide range of options for playing back and even recording video files. The `UIImageView` is not meant to be used as a video playback mechanism.

Q. ***Is there a vertical version of the slider control (UISlider)?***

A. Unfortunately, no. Only the horizontal slider is currently available in the iPhone SDK. If you want to use a vertical slider control, you'll need to implement your own.