13

Adding Images to Your View

The UIKit framework provides classes that enable you to represent and display images. In this lesson, you learn how to use the `UIImage` and `UIImageView` classes.

THE UIIMAGE CLASS

A `UIImage` object represents image data that has either been read from a file or created using Quartz primitives. Instances are immutable. Thus, their properties can't be changed once they have been created. `UIImage` instances do not provide access to the underlying image data, but do enable you to retrieve a PNG or JPEG image representation in an `NSData` object.

Images generally require large amounts of memory to store, and you should avoid creating image objects larger than 4096 x 4096 pixels. To load an image from a file into a `UIImage` object, you first need to ensure the file is in one of the formats listed in Table 13-1.

TABLE 13-1: UIImage Supported File Formats

DESCRIPTION	FILE EXTENSIONS
Portable Network Graphics	`.png`
Joint Photographic Experts Group	`.jpeg`, `.jpg`
Graphics Interchange Format	`.gif`
Windows Device Independent Bitmap	`.bmp`
Tagged Image File Format	`.tif`, `.tiff`

You also need to ensure that the file is part of the project's asset catalog. To access the asset catalog for your project, simply click on the `Assets.xcassets` file in the Project Explorer (see Figure 13-1).

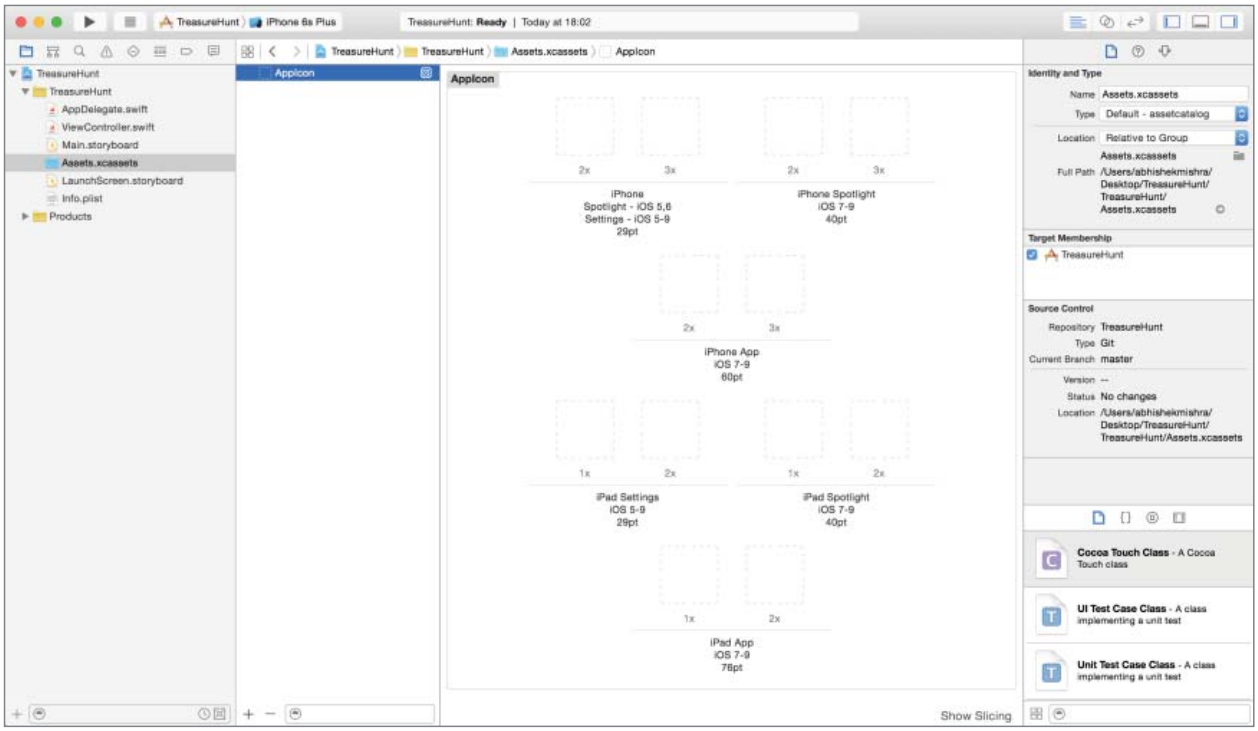

FIGURE 13-1

An asset catalog lets you keep all the images in your project in one place and access them conveniently. An asset catalog can contain the following:

- **Image sets:** An image set contains all the versions of an image, at different sizes to support different device scale factors.
- **App icons:** An app icon contains the application icon in different sizes. The application icon is used to represent the application on the iOS home screen, settings app, spotlight results, and the app store.
- **Launch images:** A launch image is a placeholder image used by iOS to stand in place of an application while the application is being loaded in the background. Once the application is loaded, iOS swaps the static launch image with the application's first screen. You will need to provide the launch image in different sizes.

Each image set in an asset catalog has a unique name that can be used to refer to the asset from both the Interface editor and code. To add a new image set to an asset catalog, select Editor ➪ New Image Set. Double-click the new image set entry within the asset catalog to rename it.

For any given image set, you must provide at least one image. It is highly recommended that you provide multiple versions at different sizes. When you create a new image asset, you can provide three sizes of the image (see Figure 13-2).

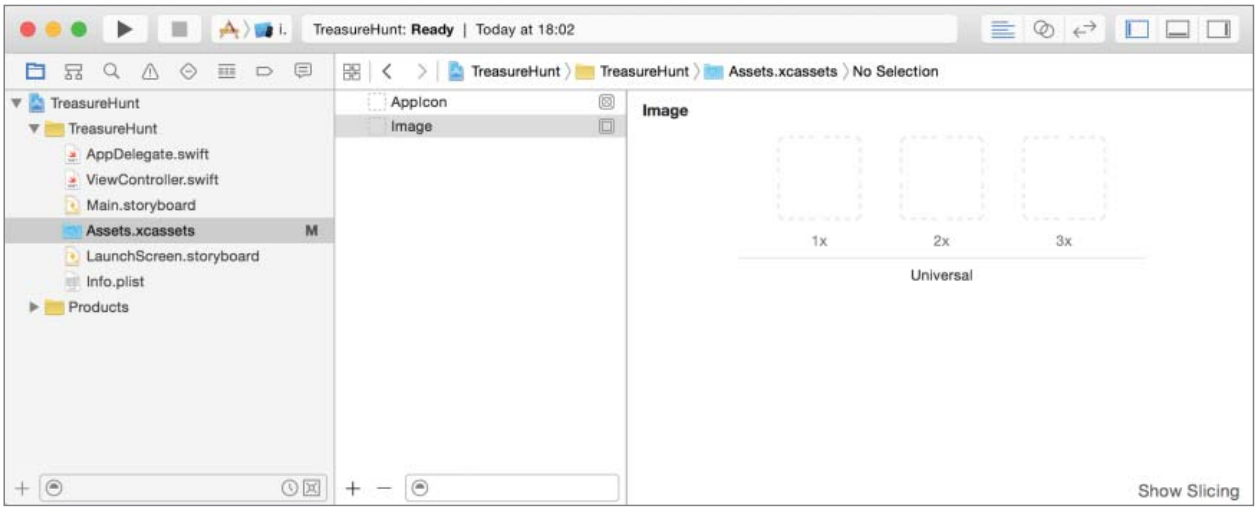

FIGURE 13-2

The base version of the image is called the 1x version, and is used when your app is running on a non-retina device. The only non-retina devices that are supported under iOS9 are the early generation iPads. To support retina devices, you provide an image that is twice the size of the base (non-retina) version. This larger image is called the 2x version. When the iPhone 6Plus was introduced with its larger screen size, a third larger image size was introduced into the mix to support this device. This larger image size, which is only used with the iPhone 6 Plus, is called the 3x version and is three times as large as the base 1x version.

Alternately, you can provide device-specific sizes by selecting Device Specific in the Devices drop-down of the Attribute Editor (see Figure 13-3).

If you have an image set called `cat` and want to load it into a `UIImage` object, you use the following code:

```
let catImage:UIImage! = UIImage(named: "cat")
```

This code uses one of the constructors of the `UIImage` class, which in turn implements an internal system cache. Thus, if you were to use this method to repeatedly load the same image file, the image data would be loaded only once and shared between the `UIImage` instances.

Loading images from your application bundle is not the only way to use `UIImage` objects. You can also create one from an online data source by downloading the data available at the URL into an `NSData` object and then instantiating a `UIImage` using an overloaded constructor that takes an `NSData` variable as input.

The following code snippet shows how to do this synchronously, but in production code, you should try and download any data from the web, including images, asynchronously. Downloading images asynchronously is an advanced topic and is not covered in this book.

```
let url = NSURL(string:"http://...")
let data = NSData(contentsOfURL: url!)
let image:UIImage! = UIImage(data: data!)
```

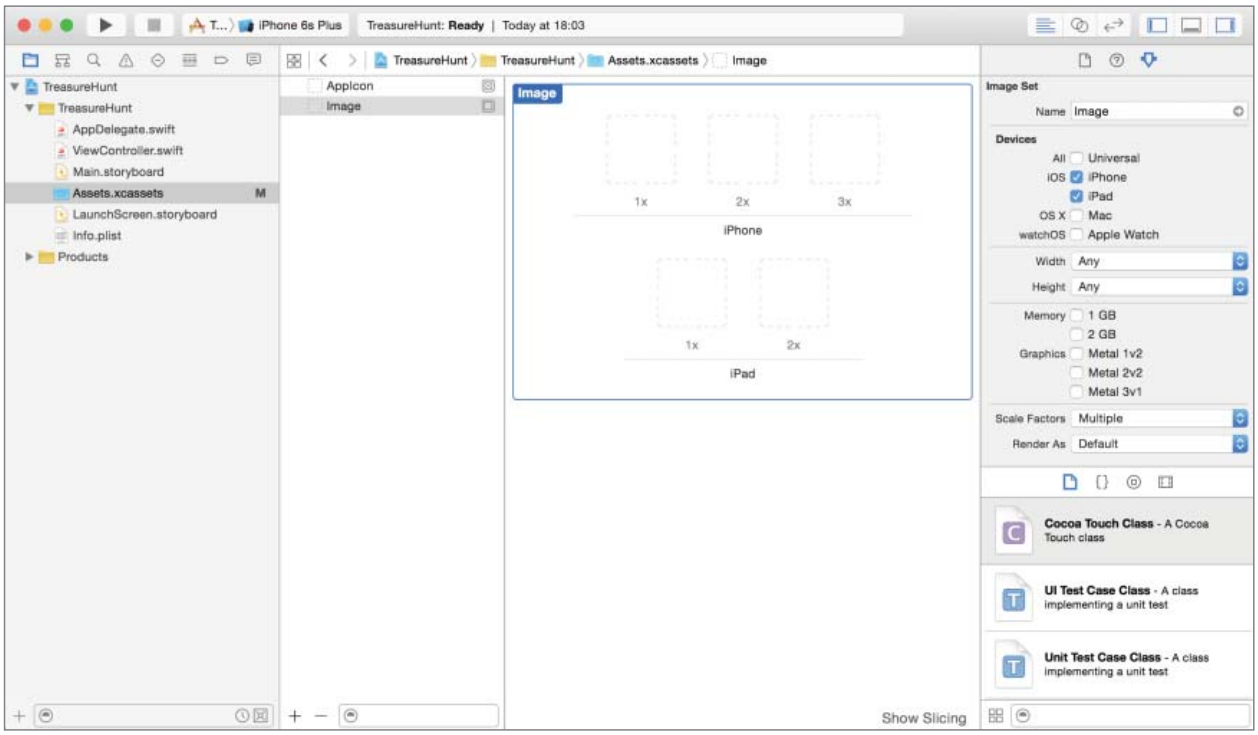

FIGURE 13-3

THE UIIMAGEVIEW CLASS

A `UIImageView` object provides a container for displaying either a single `UIImage` object or an animated series of `UIImage` objects. To add a `UIImageView` object to a view controller or storyboard scene, simply drag an Image View object from the Object library (see Figure 13-4).

To set up the default image displayed in the image view, simply select an image from the project's asset catalog for the `image` property in the Attribute inspector (see Figure 13-5).

If you wish to display a `UIImage` object in an image view programmatically, you need to create an outlet for the image view in the view controller class and set up its `image` property as follows:

```
imageView.image = UIImage(named: "cat")
```

To use a `UIImageView` object to perform simple frame animation, simply provide an array of `UIImage` objects in the image view's `animationImages` property as follows:

```
let animationImageList:[AnyObject] = [
    UIImage(named: "frame1")!,
```

```
        UIImage(named: "frame2")!,
        UIImage(named: "frame3")!,
        UIImage(named: "frame4")!
    ]

    imageView.animationImages = animationImageList
```

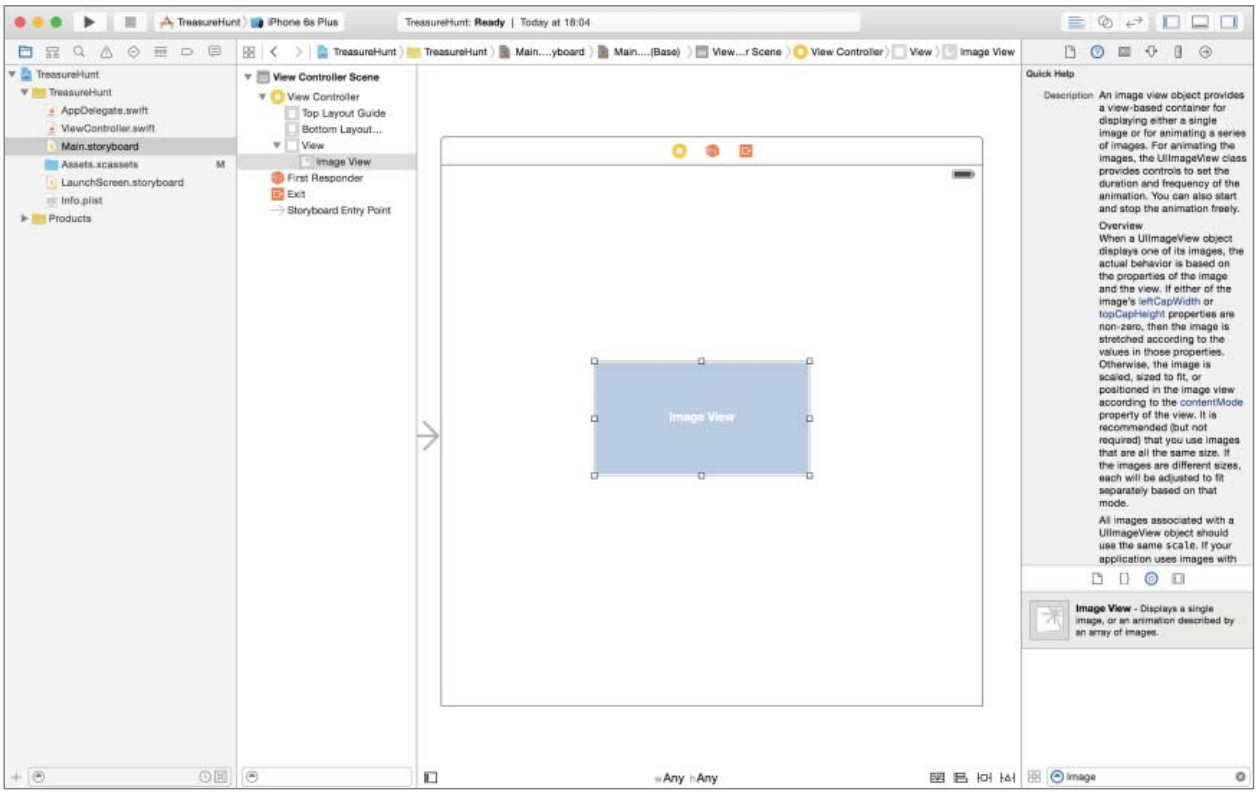

FIGURE 13-4

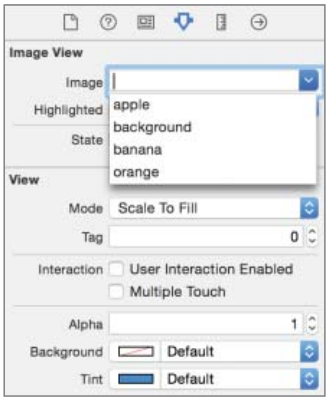

FIGURE 13-5

To kick off the animation, call the `startAnimating` method of the image view:

```
imageView.startAnimating()
```

Specify the duration of the animation in seconds, using the `animationDuration` property:

```
imageView.animationDuration = 2
```

TRY IT

In this Try It, you create a new Xcode project based on the Single View Application template called `TreasureHunt` that displays an image and asks the user to find an object in the image. When the user taps the object, a short congratulatory animation sequence is displayed.

Lesson Requirements

- Launch Xcode.
- Create a new project based on the Single View Application template.
- Edit the storyboard with Interface editor.
- Import image resources into the project.
- Add a `UILabel` instance to the default scene.
- Add two `UIImageView` instances to the default scene.
- Use a gesture recognizer to detect a tap on the image and display an alert view.
- If the tap occurs over a specific region of the image, display a congratulatory frame animation.

> **REFERENCE** *The code for this Try It is available at* `www.wrox.com/go/swiftios`.

Hints

- To show the Object library, select View ➪ Utilities ➪ Show Object Library.
- To show the assistant editor, select View ➪ Assistant Editor ➪ Show Assistant Editor.

Step-by-Step

- Create a Single View Application in Xcode called `TreasureHunt`.
  1. Launch Xcode and create a new application by selecting File ➪ New ➪ Project.
  2. Select the Single View Application template from the list of iOS project templates.

3. In the project options screen, use the following values:
   - **Product Name:** TreasureHunt
   - **Organization Name:** your company
   - **Organization Identifier:** com.yourcompany
   - **Language:** Swift
   - **Devices:** iPhone
   - **Use Core Data:** Unchecked
   - **Include Unit Tests:** Unchecked
   - **Include UI Tests:** Unchecked
4. Save the project onto your hard disk.

- Add image resources to your project.
  1. Ensure the project navigator is visible. To show it, select View ➪ Navigators ➪ Show Project Navigator.
  2. Open the `Assets.xcassets` file by clicking on it in the project navigator.
  3. Navigate to the `Images` folder in this chapter's resources from the website.
  4. Create a new image set by selecting Editor ➪ New Image Set and name this new image set `beads`.
  5. Drag the `beads1x.png`, `beads2x.png`, and `beads3x.png` images from this chapter's resources into the appropriate placeholders in the image set.
  6. Create a new Image set by selecting Editor ➪ New Image Set and name this new image set `animframe1`.
  7. Drag the `animframe1_1x.png`, `animframe1_2x.png`, and `animframe1_3x.png` images from this chapter's resources into the appropriate placeholders in the image set.
  8. Similarly, create new image sets called `animframe2`, `animframe3`, `animframe4`, `animframe5`, and `animframe6`, and use the appropriate images from this chapter's resources folder.
- Add a `UILabel` instance to the default scene.
  1. Open the `MainStoryboard.storyboard` file in Interface Builder.
  2. Ensure the Object library is visible. To show it, select View ➪ Utilities ➪ Show Object Library.
  3. From the Object library, drag and drop a Label object onto the scene.
  4. Use the Attribute inspector to set the Text attribute of the label to `Tap the blue bead!` To show the Attribute inspector, select View ➪ Utilities ➪ Show Attributes Inspector.

5. Select the label in the scene, and select Editor ➪ Size to Fit Contents to ensure the label is large enough to show its contents.
6. Select the label in the scene and click the Align button to display the alignment constraint editor. Add a constraint to center the label horizontally.
7. Select the label in the scene and click the Pin button to display the constraints editor. Ensure the Constrain to margins options is unchecked and set the following constraint:
    - ➤ The distance from the top of the label to the view should be 10.
8. Update the frames to match the constraints you have set.
    - ➤ Click on the View controller item in the dock above the storyboard scene. This is the first of the three icons located directly above the selected storyboard scene.
    - ➤ Select Editor ➪ Resolve Auto Layout Issues ➪ Update Frames.

➤ Add two `UIImageView` instances to the default scene.

1. From the Object library, drag and drop an Image View object onto the scene, and place it below the label.
2. Use the Attribute inspector to set the Image attribute of the image view to `bead`. To show the Attribute inspector, select View ➪ Utilities ➪ Show Attributes Inspector.
3. Using the Attribute inspector, set the View Mode attribute to Aspect Fill.
4. Select the image view in the scene, and select Editor ➪ Size to Fit Contents to ensure the image view is large enough to show its image.
5. Select the image view in the scene and click the Align button to display the alignment constraint editor. Add a constraint to center the image view horizontally.
6. Select the image view in the scene and click the Pin button to display the constraints editor. Ensure the Constrain to margins options is unchecked and set the following constraint:
    - ➤ The vertical distance between the label and the image view should be 10.
7. Update the frames to match the constraints you have set.
    - ➤ Click on the View controller item in the dock above the storyboard scene. This is the first of the three icons located directly above the selected storyboard scene.
    - ➤ Select Editor ➪ Resolve Auto Layout Issues ➪ Update Frames.
8. Use the assistant editor to create an outlet in the view controller class called `largeImage` and connect the image view to it.
9. From the Object library, drag and drop a second Image View object to the scene.

10. Use the Attribute inspector to set the Image attribute of the image view to `animframe1`. To show the Attribute inspector, select View ➪ Utilities ➪ Show Attributes Inspector.
11. Using the Attribute inspector, set the View Mode attribute to Aspect Fill.
12. Select the image view in the scene and click the Align button to display the alignment constraint editor. Add a couple of constrains to center the image view horizontally and vertically.
13. Update the frames to match the constraints you have set.
14. Use the Assistant editor to create an outlet in the view controller class called `animated Image` and connect the image view to it.

➤ Add a tap gesture recognizer and use it to show an animated image sequence when the blue bead is tapped. Gesture recognizers are covered in detail in Lesson 21.

1. Update the `viewDidLoad` method of the view controller class to resemble the following:

```
override func viewDidLoad() {

super.viewDidLoad()

// install tap gestue recognizer.
let tapRecognizer = UITapGestureRecognizer(target: self,
                    action:"handleTap:")

tapRecognizer.cancelsTouchesInView = false
self.view.addGestureRecognizer(tapRecognizer)

// setup animatedImage
let frameArray:[UIImage] = [
    UIImage(named: "animframe1")!,
    UIImage(named: "animframe2")!,
    UIImage(named: "animframe3")!,
    UIImage(named: "animframe4")!,
    UIImage(named: "animframe5")!,
    UIImage(named: "animframe6")!
]

animatedImage.animationImages = frameArray
animatedImage.animationDuration = 0.5
animatedImage.animationRepeatCount = 1
animatedImage.userInteractionEnabled = false
animatedImage.hidden = true
}
```

2. Add the following method to the `ViewController.swift` file:

```
func handleTap(sender:UITapGestureRecognizer) {

    let startLocation:CGPoint =
```

```
        sender.locationInView(self.largeImage)

        let scaleFactor = self.largeImage.frame.size.height / 430.0;

        if ((startLocation.y >= 211 * scaleFactor) &&
            (startLocation.y <= (211 + 104) * scaleFactor))
        {
            animatedImage.hidden = false
            animatedImage.startAnimating()
        }
    }
```

➤ Test your app in the iOS Simulator.

Click the Run button in the Xcode toolbar. Alternatively, you can select Project ➪ Run.