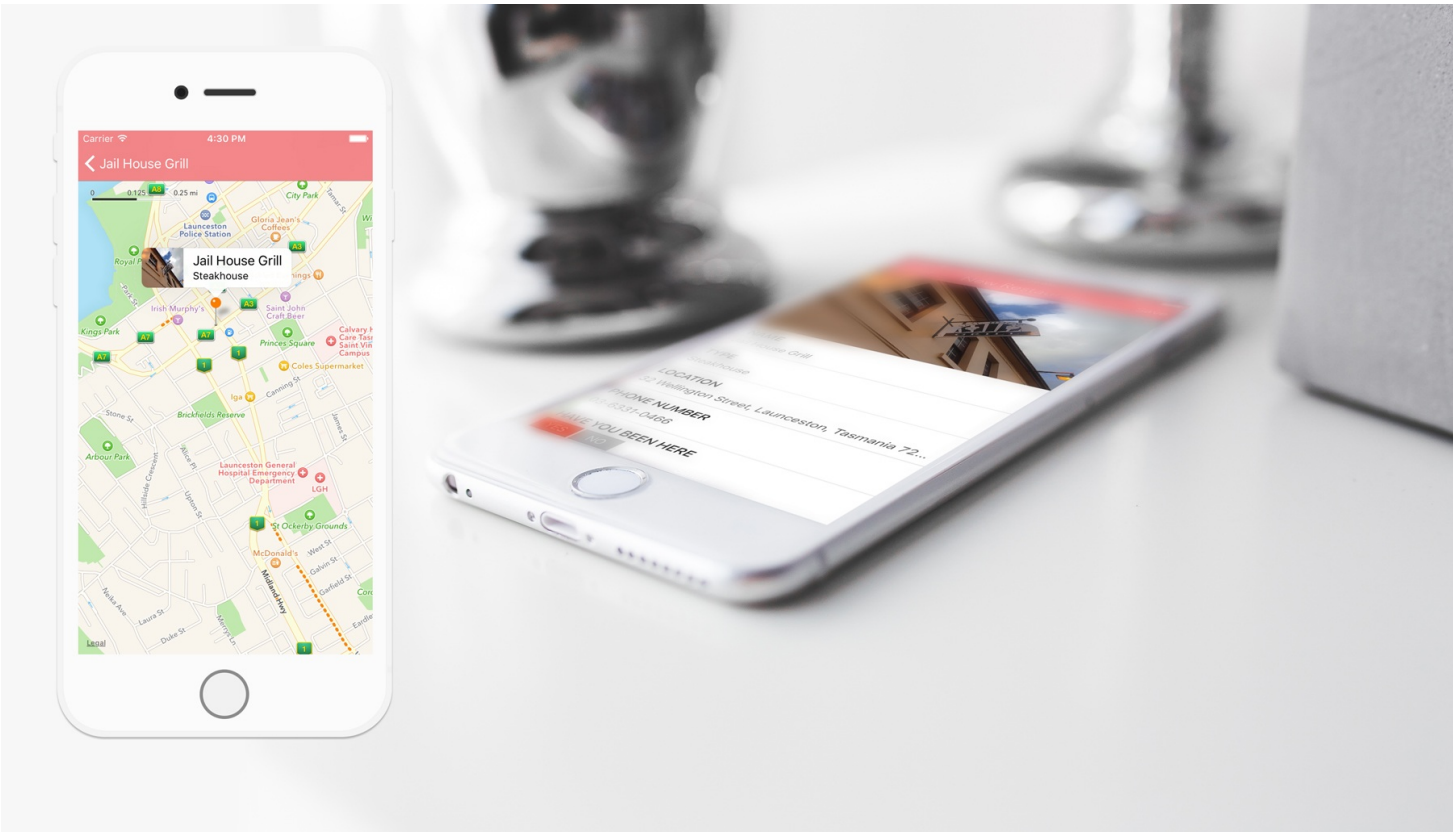
Chapter 17
Working with Maps



> The longer it takes to develope, the less likely it is to launch.
>
> -Jason Fried, Basecamp

The MapKit framework provides APIs for developers to display maps, navigate through maps, add annotations for specific locations, add overlays on existing maps, etc. With the framework you can embed a fully functional map interface into your app without any coding.

In iOS 9, the MapKit framework brings us a number of new features such as pin customization, transit and flyover support. We will go over some new features with you. In particular, you will learn a few things about the framework:

- How to embed a map

- How to translate an address into coordinates using Geocoder
- How to add and customize a pin (i.e. annotation) on map
- How to customize an annotation

To give you a better understanding of the MapKit framework, we will add a map feature in the FoodPin app. After the change, the app will bring up a map when a user taps the address of a restaurant, and pin its location on the map.

Cool, right? It's gonna be fun. Let's get started.

Using MapKit Framework

By default, the MapKit framework is not bundled in the Xcode project. To use it, you have to first add the framework and bundle it in your project. But you don't need to do it manually. Xcode has a capability section that lets you configure frameworks for various Apple technologies such as Maps and iCloud.

In the project navigator, select the FoodPin project and then select the FoodPin target. You can then enable the `Maps` feature under the Capabilities section. Just flip the switch to ON, and Xcode automatically configures your project to use the MapKit framework.

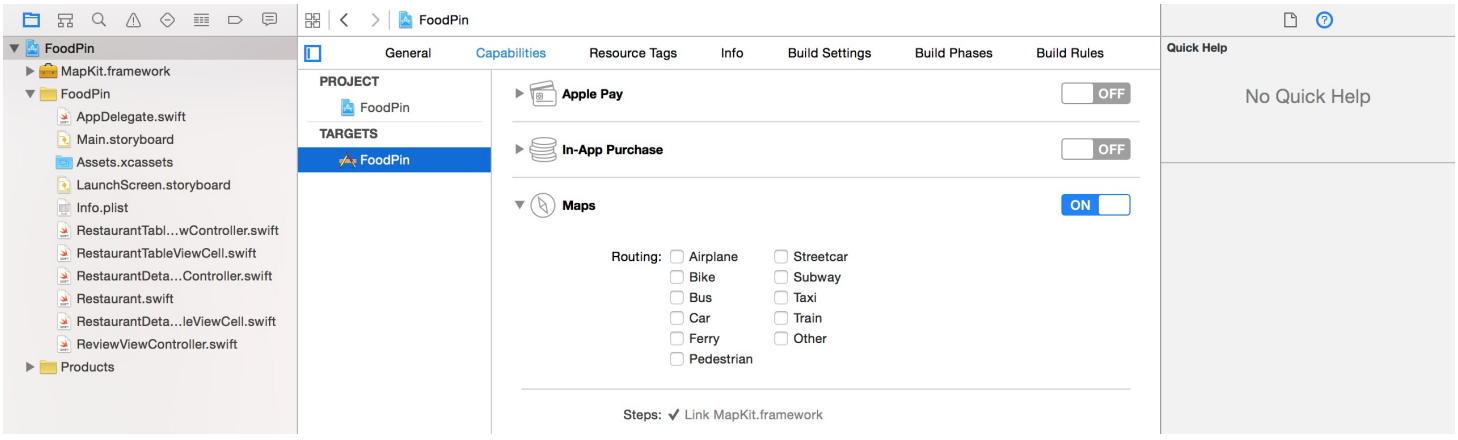


Figure 17-1. Enabling Maps in your Xcode project

Adding a Map Interface to Your App

What we're going to do is to add a map button next to the address in the Detail View. When a user taps the map icon, the app navigates to a map view controller showing the restaurant location. First, download the map icons from https://www.dropbox.com/s/mh007pm81ao8qa9/mapicon.zip?dl=0 and add the icons to `Assets.xcassets`.

```
The map icons is made by SimpleIcon from www.flaticon.com and is licensed by CC BY 3.0
```

Now open `Main.storyboard` and drag a button from the Object library to the image view of the Detail View Controller. In the Attributes inspector, set the button's image to `map`. Change its size to 40 by 40 points. If you like, change its *background* color to `red`, and *tint* to `white`. Next, add a few layout constraints for the map button. In the layout bar, click the *Pin* button and add the necessary constraints (refer to figure 17-2).

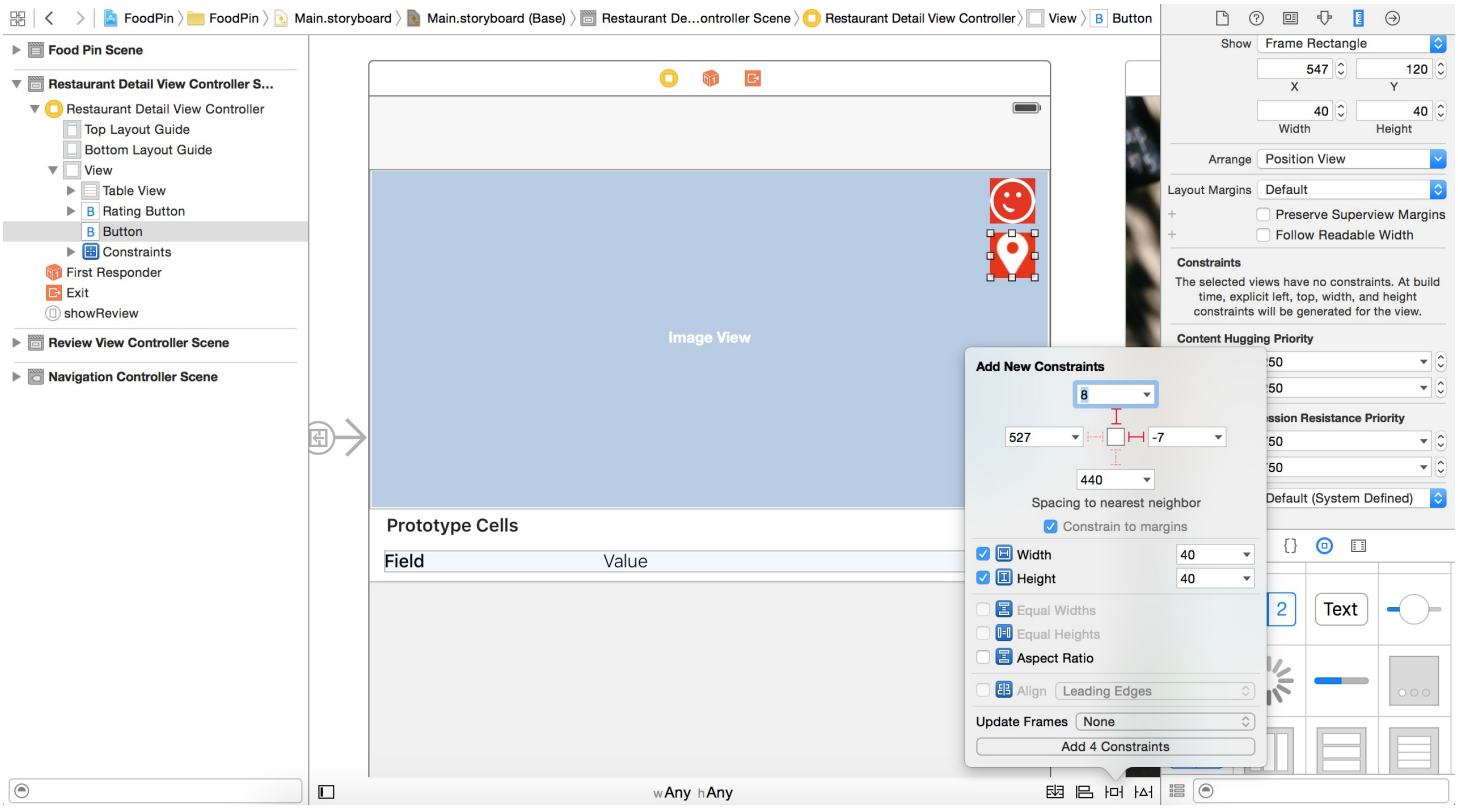


Figure 17-2. Adding a map button to the image view

To bring up a map when the map button is tapped, we first need to create a new view

controller. Drag a view controller object from the Object library to the storyboard, and then add the Map Kit View object to the view controller. Again, you can let Interface Builder add the layout constraints for you. Resize the map view to fit the view, and then click "Resolve Auto Layout Issues" > "Add Missing Constraints".

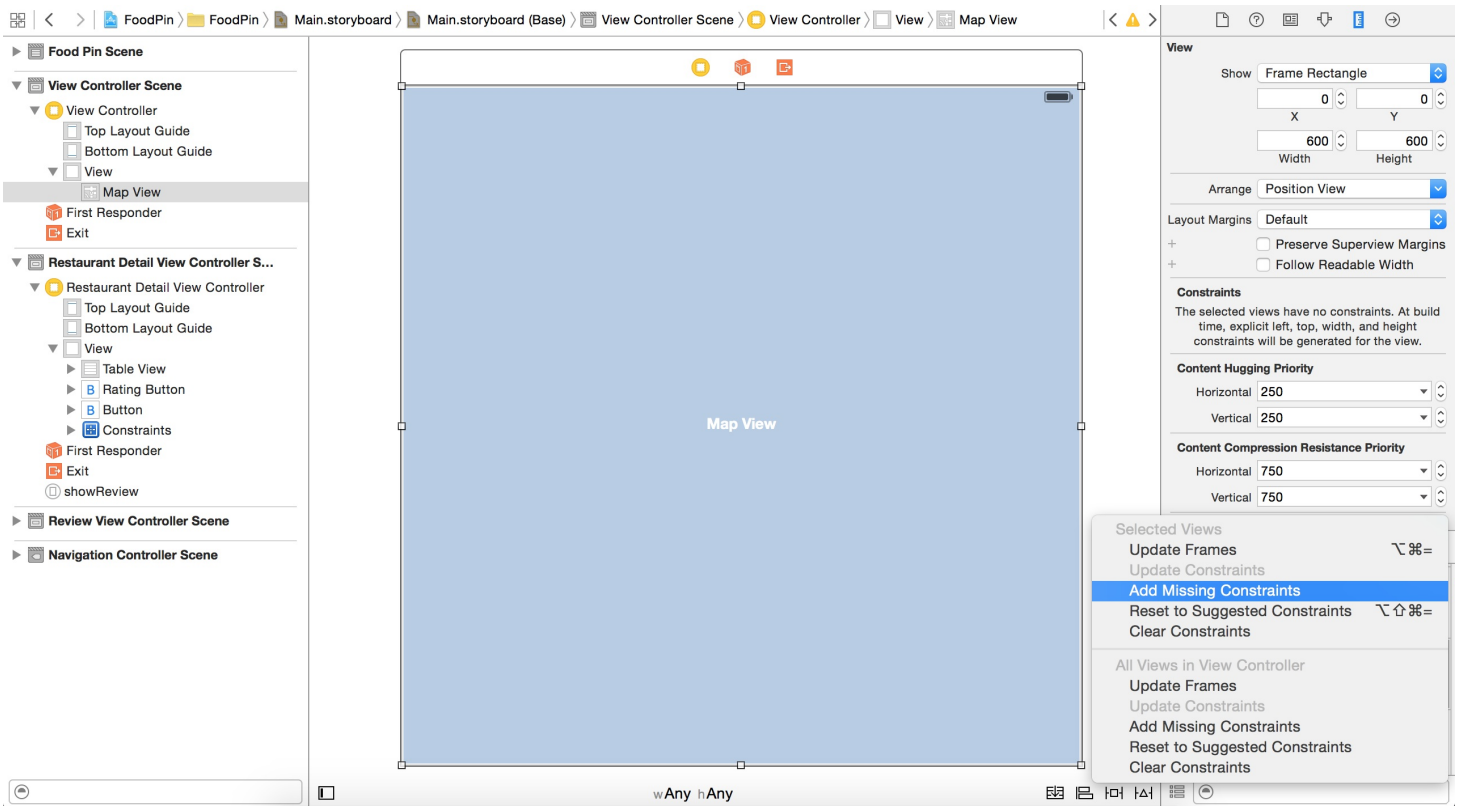


Figure 17-3. Adding a map view

Now hold control key and drag from the *Map* button to the new map view controller to create a segue. Select *show* as the segue type. For the segue we just created, set the identifier to `showMap` under the Attribute inspector.

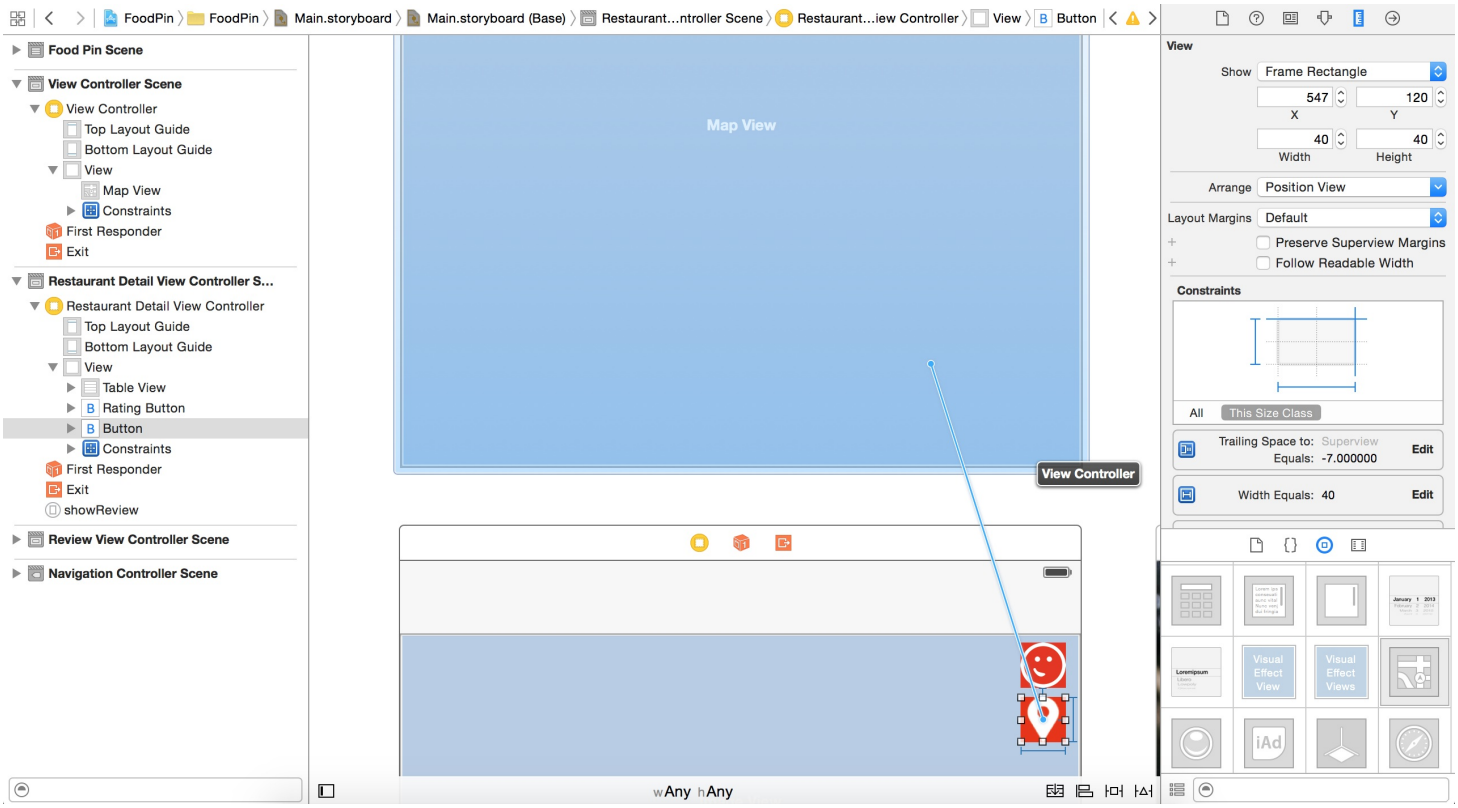


Figure 17-4. Defining layout constraints for the map view

If you compile and run the app, tap the map button in the detail view. The app should bring up a fully functional map. This is the power of the MapKit framework. Without a line of code, you already embed a map within your app.

The map view provides various options such as zooming and scrolling for customizing the map features. In Interface Builder, select the map view and go to the Attributes inspector. You can edit the options, and change the map type from Standard to Satellite or Hybrid. Optionally you can enable *User Location* to display the current location of the user. You're free to play around with these options to see how they work. Of course, we're not done yet. For now, the app only displays a default map. We will need to write some code to pin the location of a restaurant on map.

Converting an Address into Coordinates Using Geocoder

To highlight a location on the map, you cannot just use a physical address. The MapKit framework doesn't work like that. Instead, the map has to know the geographical coordinates

expressed in terms of the latitude and longitude of the corresponding point on the globe.

The framework provides a `Geocoder` class for developers to convert a textual address, known as *placemark*, into global coordinates. This process is usually referred to *forward geocoding*. Conversely, you can use `Geocoder` to convert latitude and longtitude values back to a *placemark*. This process is known as *reverse geocoding*.

To initiate a forward-geocoding request using the `CLGeocoder` class, all you need do is create an instance of `CLGeocoder`, followed by calling the `geocodeAddressString` method with the address parameter. Here is an example:

```
let geoCoder = CLGeocoder()
geoCoder.geocodeAddressString("524 Ct St, Brooklyn, NY 11231",
completionHandler: { placemarks, error in

// Process the placemark

})
```

There is no designated format of an address string. The method submits the specified location data to the geocoding server asynchronously. The server then parses the address and returns you an array of placemark objects. The number of placemark objects returned greatly depends on the address you provide. The more specific the address information you have given, the better the result. If your address is not specific enough, you may end up with multiple placemark objects.

With the `placemark` object, which is an instance of `CLPlacemark` class, you can easily get the geographical coordinate of the address using the code below:

```
let coordinate = placemark.location.coordinate
```

The completion handler is the code block to be executed after the forward-geocoding request completes. Operations like annotating the placemark will be done in the code block.

Adding Annotations to a Map

Now that you have a basic idea of `Geocoder` and understand how to get the global coordinates of an address, we will look at how you can pin a location on maps. To do that, the MapKit

framework provides an annotation feature for you to pinpoint a specific location.

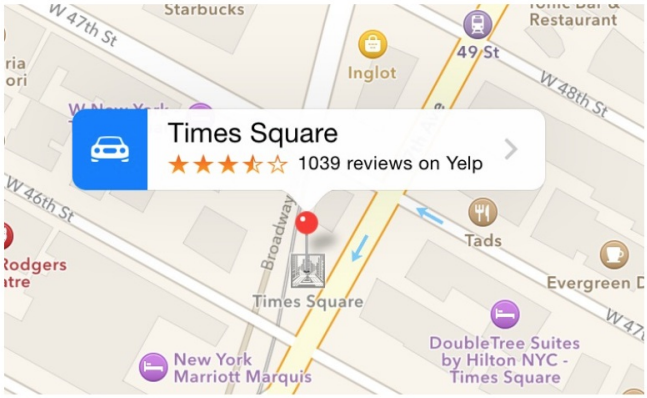


Figure 17-5. An annotation

An annotation can appear in many forms. The pin annotation that you usually see in a Maps app is an example of annotations. Typically an annotation consists of an image (e.g. pin) and a callout bubble that displays additional information about the location.

From a developer's point of view, an annotation actually consists of two different objects:

- an annotation object - which stores the data of an annotation such as the name of the placemark. The object should conform to the `MKAnnotation` protocol as defined in the Map Kit.
- an annotation view - which is the actual object for the visual representation of the annotation. The pin image is an example. If you want to display the annotation in your own form (say, use pencil instead of pin), you'll need to create to your own annotation view.

The MapKit framework comes with a standard annotation object and an annotation view. So you do not need to create your own, unless you want to customize the annotation.

In general, to add a standard annotation to a map, here is the code snippet you need:

```
let annotation = MKPointAnnotation()
annotation.title = "Times Square"
annotation.coordinate = placemark.location.coordinate
```

```
mapView.showAnnotations([annotation], animated: true)
mapView.selectAnnotation(annotation, animated: true)
```

The `MKPointAnnotation` class is standard class, which adopts the `MKAnnotation` protocol. By specifying the coordinates in the annotation object, you can call the `showAnnotations` method of the `mapView` object to put a pin on the map. The method is smart enough to determine the best fit region for the annotation. By default, the callout bubble is not shown on the map until the user taps the pin. If you want to display the bubble without user interaction, you can invoke the `selectAnnotation` method.

Adding an Annotation to the FoodPin App

After introducing you the basics of annotations and geocoding, let's get back to the FoodPin project. As usual, we first create a custom class for the map view controller. In the project navigator, right click the `FoodPin` folder and select "New File...". Create the new class using the `Cocoa Touch class` template and name the class `MapViewController`. Make sure you set it as a subclass of `UIViewController`, and save the file.

Recalled that we have configured the MapKit framework in the project, you have to add an `import` statement in order to use it. Insert the following line of code at the very beginning of the `MapViewController.swift` file:

```
import MapKit
```

Next, declare the following outlet variable for the map view and another variable for the selected restaurant:

```
@IBOutlet var mapView:MKMapView!

var restaurant:Restaurant!
```

The outlet variable is used for establishing a connection with the map view in the storyboard. Go to Interface Builder and select the map view controller. Under the Identity inspector, set the custom class to `MapViewController`. Then establish a connection between the outlet variable and the map view.

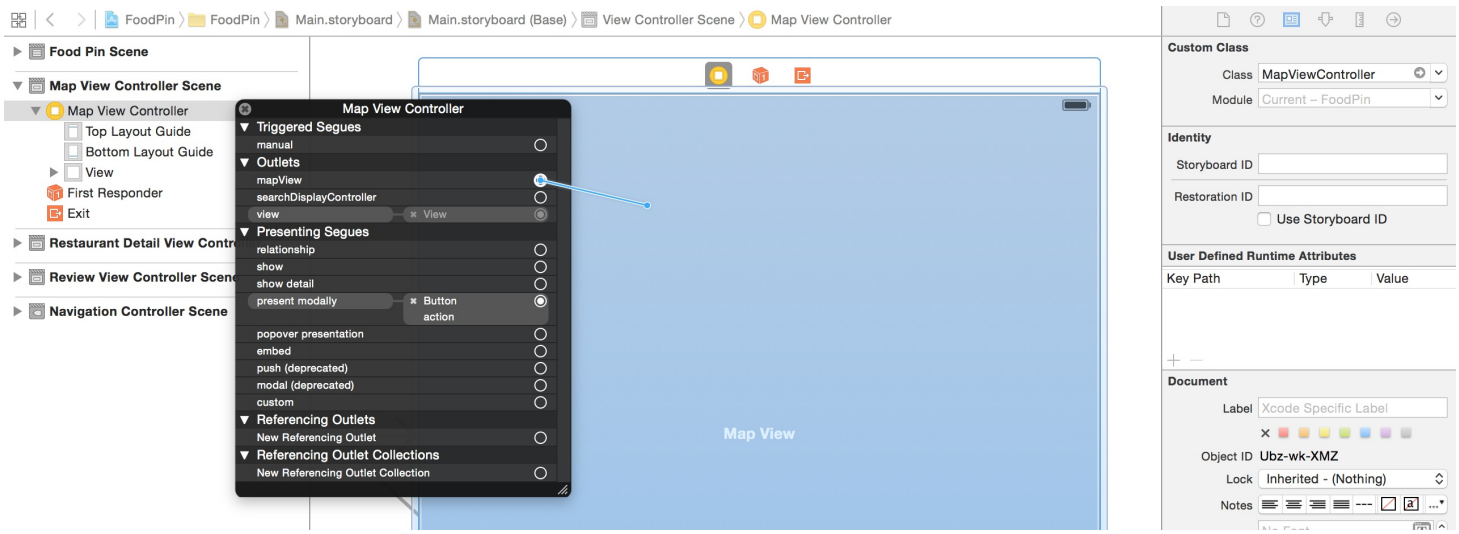


Figure 17-6. Establish a connection between MKMapView and the outlet variable

To add an annotation on map, update the `viewDidLoad` method to the following code:

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Convert address to coordinate and annotate it on map
    let geoCoder = CLGeocoder()
    geoCoder.geocodeAddressString(restaurant.location, completionHandler: {
placemarks, error in
        if error != nil {
            print(error)
            return
        }

        if let placemarks = placemarks {
            // Get the first placemark
            let placemark = placemarks[0]

            // Add annotation
            let annotation = MKPointAnnotation()
            annotation.title = self.restaurant.name
            annotation.subtitle = self.restaurant.type

            if let location = placemark.location {
                annotation.coordinate = location.coordinate

                // Display the annotation
                self.mapView.showAnnotations([annotation], animated: true)
                self.mapView.selectAnnotation(annotation, animated: true)
```

```
            }
        }

    })
}
```

I'll not go into the above code line by line as we've discussed the usage of `Geocoder` and annotation earlier. In brief, we first convert the address of the selected restaurant (i.e. `restaurant.location`) into coordinates using `Geocoder`. In most cases, the placemarks array should contain a single entry. So we just pick the first element from the array, followed by displaying the annotation on the map view.

There is still one thing left before testing the app. We haven't passed the selected restaurant to the map view controller. In the `DetailViewController` class, add the `prepareForSegue` method:

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject!) {
    if segue.identifier == "showMap" {
        let destinationController = segue.destinationViewController as!
MapViewController
        destinationController.restaurant = restaurant
    }
}
```

We simply get the selected restaurant and pass it to the destination view controller. In this case, it's the `MapViewController` class. Okay, let's compile and run the app. Tap the Map button in the detail view and you'll see a pin on the map.

Figure 17-7. Tapping the map button now shows a map with the restaurant's location

Adding an Image to the Annotation View

Wouldn't it be great if we can show the restaurant thumbnail in the callout bubble?

As mentioned in the beginning of this chapter, an annotation view controls the visual part of an annotation. To add a thumbnail or image in the annotation, you have to modify the annotation view. And, to do that, you have to adopt the `MKMapViewDelegate` protocol, which defines a set of optional methods that you can use to receive map-related update messages. The map view uses one of these methods to request annotation. Every time when the map view needs an annotation view, it calls the `mapView(_:viewForAnnotation:)` method:

```
optional func mapView(_ mapView: MKMapView!, viewForAnnotation annotation:
```

```
MKAnnotation!) -> MKAnnotationView!
```

So far we haven't adopted the protocol and provide our own implement for the method. This is why a default annotation view is displayed. We're going to implement the method and create our own annotation view that embeds the restaurant image.

First, open the `MapViewController.swift` file and adopt the `MKMapViewDelegate` protocol:

```
class MapViewController: UIViewController, MKMapViewDelegate
```

In the `viewDidLoad` method, add the following line of code to define the delegate of mapView:

```
mapView.delegate = self
```

Quick note: Here we define MapViewController as the delegate object of mapView. The delegation pattern is among the most common patterns in iOS development. By now, I assume you have a good understanding of the delegate pattern. If not, read chapter 7 again.

Next implement the `mapView(_:viewForAnnotation:)` method using the following code:

```
func mapView(mapView: MKMapView, viewForAnnotation annotation: MKAnnotation) ->
MKAnnotationView? {
    let identifier = "MyPin"

    if annotation.isKindOfClass(MKUserLocation) {
        return nil
    }

    // Reuse the annotation if possible
    var annotationView:MKPinAnnotationView? =
mapView.dequeueReusableAnnotationViewWithIdentifier(identifier) as?
MKPinAnnotationView

    if annotationView == nil {
        annotationView = MKPinAnnotationView(annotation: annotation,
reuseIdentifier: identifier)
        annotationView?.canShowCallout = true
    }

    let leftIconView = UIImageView(frame: CGRectMake(0, 0, 53, 53))
    leftIconView.image = UIImage(named: restaurant.image)
    annotationView?.leftCalloutAccessoryView = leftIconView

    return annotationView
```

```
}
```

Let's go through the above code line by line.

Other than placemark, the user's current location is also a kind of annotation. The map view will also call this method when annotating the user's location. As you may know, the current location is displayed as a blue dot in maps. Even though we haven't enabled the app to display the current location, we don't want to change its annotation view. At the very beginning of the code, we verify if the annotation object is a kind of `MKUserLocation`. If yes, we simply return `nil` and the map view will keep displaying the location using a blue dot.

For performance reasons, it is preferred to reuse an existing annotation view instead of creating a new one. The map view is intelligent enough to cache unused annotation views that it isn't using. Similar to `UITableView`, we can call up the `dequeueReusableAnnotationViewWithIdentifier` method to see if any unused views are available. We then downcast it to `MKPinAnnotationView`.

If there are no unused views available, we create a new one by instantiating a `MKPinAnnotationView` object with `the canShowCallout` property set to `true`. We still use the standard pin as the annotation view. But we add a thumbnail of the selected restaurant to `leftCalloutAccessoryView`, which is a view shown on the left side of the callout bubble.

Cool! We're done. Press the Run button and launch the app. Pick a restaurant and tap the Map button. You should see a thumbnail in the callout bubble.

Figure 17-8. An annotation with a thumbnail

Customizing the Pin Color

In iOS 9, you can customize the color of the pin to any color you want. Apple introduces a new property called *pinTintColor* for `MKPinAnnotationView`. To change the pin color, all you need to do is assign the *pinTintColor* property with a `UIColor` object.

```
annotationView?.pinTintColor = UIColor.orangeColor()
```

Figure 17-9. Changing the pin color of the annotation

Map Customizations

iOS 9 also introduces a number of handy properties to the `MKMapView` class that lets developers control what goes on the map view. Here are the three new properties for you to control the content of a map view:

- showTraffic - shows any high traffic on your map view
- showScale - shows a scale on the top-left corner of your map view
- showCompass - displays a compass control on the top-right corner of your map view

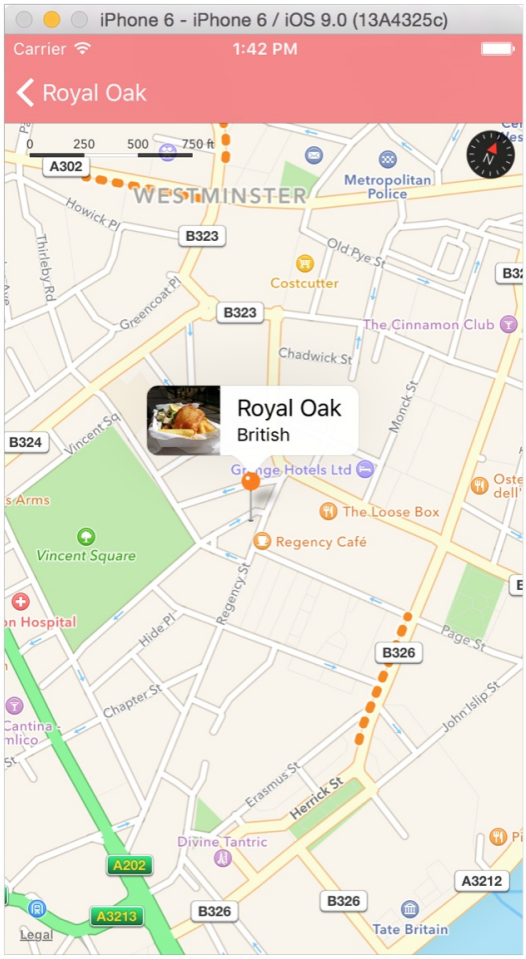


Figure 17-10. Show compass, scale and traffic on your map view

As a demo, you can insert the following lines of code in the `viewDidLoad` method to give it a try:

```
mapView.showsCompass = true
mapView.showsScale = true
mapView.showsTraffic = true
```

Summary

In this chapter, I've walked you through the basics of the MapKit framework. By now, you should know how to embed a map in your app and add an annotation. But this is just a beginning. There is a lot more you can do from here. One thing you can do is further explore MKDirection . It provides you with route-based directions data from Apple servers to get

travel-time information or driving or walking directions. You can take the app further by showing directions.

For your reference, you can download the complete Xcode project from https://www.dropbox.com/s/4r6do8gll9em234/FoodPinMaps.zip?dl=0.