



Build Location Apps on iOS with Swift

Use Apple Maps, Google Maps, and Mapbox to Code Location Aware Mobile Apps

Jeffrey Linwood

Apress®

Build Location Apps on iOS with Swift

**Use Apple Maps, Google Maps,
and Mapbox to Code Location
Aware Mobile Apps**

Jeffrey Linwood

Apress®

Build Location Apps on iOS with Swift: Use Apple Maps, Google Maps, and Mapbox to Code Location Aware Mobile Apps

Jeffrey Linwood
Austin, TX, USA

ISBN-13 (pbk): 978-1-4842-6082-1
<https://doi.org/10.1007/978-1-4842-6083-8>

ISBN-13 (electronic): 978-1-4842-6083-8

Copyright © 2020 by Jeffrey Linwood

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Aaron Black
Development Editor: James Markham
Coordinating Editor: Jessica Vakili

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-6082-1. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

Table of Contents

About the Author	ix
About the Technical Reviewer	xi
Chapter 1: Creating Your First MapKit App.....	1
Getting started	1
Adding a map.....	5
Adding a pin to your map	10
Summary.....	16
Chapter 2: Getting the User's Location	17
Privacy and location permissions	17
Location permissions in the Info.plist file.....	18
Requesting location permissions from the end user	19
Requesting location updates	22
Displaying the user's location on the map	24
Summary.....	26
Chapter 3: Displaying Annotations on a MapKit Map	27
Understanding MapKit and annotations.....	27
Using a custom annotation class	28
Display custom annotations.....	29
Customizing pins for annotations	30
Handling the user location	33
Reusing annotation views.....	33

TABLE OF CONTENTS

Dequeuing and creating annotation views.....	35
Setting images on annotation views.....	36
Using callouts with annotations.....	38
Summary.....	39
Chapter 4: Searching for Points of Interest	41
Getting started with local search	41
Exploring the map items in the response	44
Displaying the search results on a map.....	44
Creating annotations for results.....	47
Filtering with points of interest categories	47
Chapter 5: Getting Directions with MapKit	53
Understanding the Directions API	53
Getting started with directions.....	54
Understanding route steps.....	61
Building step-by-step directions.....	62
Displaying the current step	64
Saving the route in an instance variable	66
Actions for the previous and next buttons.....	67
Next steps	73
Summary.....	74
Chapter 6: Working with Geofences in CoreLocation.....	75
Concepts for region monitoring	75
Setting up your location-based application	76
Getting started with region monitoring	78
Listening for geofence triggers.....	80

Displaying a local notification inside the app	82
Setting up the notification center	82
Displaying a local notification for a region	84
Monitoring region changes in the app delegate	85
Removing geofences from the app	87
Chapter 7: Displaying a Map with the Google Maps SDK	89
Using the Google Maps Platform.....	89
Installing the Google Maps SDK for iOS library	90
Setting up a Google Maps API Key	93
Including the API Key in your application.....	98
Displaying Google Maps with a map view	99
Allowing the Google Maps and Chrome URL schemes	102
Changing display options on the map view	103
Chapter 8: Exploring Google Map Views.....	107
Changing the type of map tiles	107
Displaying map markers	108
Changing the marker icon	110
Responding to marker events	110
User data and markers	111
Adding shapes to the map	112
Circles.....	113
Polylines and paths	114
Polygons.....	117
Removing markers and shapes	119
Conclusion	119

TABLE OF CONTENTS

Chapter 9: Using Directions with the Google Directions API	121
Setting up the Google Directions API	122
Restricting the API Key	124
Using the Google Directions API	126
Creating the URL	127
Calling the Directions API with URLSession	128
Processing the directions response	130
Displaying the route as a polyline	135
Updating the map bounding box	136
Next steps: Displaying each leg and step	138
Chapter 10: Using Google Places in Your iOS App	139
Building a places finder with a map	139
Creating the project	140
Getting a Google Places API key	140
Setting up CocoaPods	145
Providing an API key for Google Places and Google Maps	147
Creating the user interface	148
Understanding the autocomplete search	151
Creating the autocomplete view controller	152
Setting geographic boundaries for the search	152
Requesting a subset of data fields	153
Filtering results by type	155
Displaying the view controller	156
Implementing the delegate for autocomplete	158
Displaying the place on the map	160
Additional functionality in Places SDK	162

Chapter 11: Getting Started with the Mapbox SDK.....	165
Getting a Mapbox access token.....	165
Starting a new project with the Mapbox SDK.....	167
Displaying a map view.....	170
Customizing the map view's appearance.....	172
Display a point annotation on the map.....	174
Chapter 12: Customizing Map Styles with Mapbox.....	179
Getting ready for map styles.....	179
Changing the default style on the map view.....	180
Creating map styles with Mapbox Studio.....	180
Displaying the new map style in the app.....	191
Chapter 13: Working with Datasets in Mapbox Studio.....	193
Earthquake map.....	193
Downloading the earthquake data.....	194
Creating a dataset on Mapbox Studio.....	197
Creating a tileset.....	204
Adding the tileset to a style.....	206
Styling the features with Mapbox Studio.....	208
Displaying the dataset in the app.....	215
Detecting a tap on the features.....	217
Conclusion.....	218
Chapter 14: Turn-by-Turn Navigation with Mapbox.....	221
Setting up your app project.....	222
Setting up CocoaPods.....	222
Adding entries to Info.plist.....	223
Adding required capabilities.....	223

TABLE OF CONTENTS

Using the Mapbox Directions API	225
Displaying the navigation user interface	228
Using the Mapbox simulated navigation	231
Customizing the navigation experience	233
Conclusion	234
Chapter 15: Using Offline Maps with Mapbox	235
Setting up your app project.....	235
Understanding offline map downloading	236
Estimating the number of tiles used	237
Downloading an offline map pack	238
Monitoring offline map pack downloads.....	242
Considerations for using offline map tiles	247
Index.....	249

About the Author

Jeffrey Linwood is an experienced software developer who has worked on many iOS and Android apps that use maps or location functionality. He's also taught and mentored college student application teams as they develop their first iOS apps. While teaching, he noticed a lack of good sample applications and tutorials for map and location applications. Jeff also enjoys running, hiking, and spending time with his wife, Clover, in Austin, Texas. You can follow Jeff on Twitter at @jefflinwood, or on his blog, <https://www.jefflinwood.com>.

About the Technical Reviewer

Felipe Laso is a Senior Systems Engineer working at Lextech Global Services. He's also an aspiring game designer/programmer. You can follow him on Twitter at @iFeliLM or on his blog.

CHAPTER 1

Creating Your First MapKit App

This book will be project based – starting out simple and then getting more complicated. With that in mind, our first iOS app will only be one screen that displays a map. That map will have one pin on it, with the location of my city, Austin, Texas. Feel free to use your own town for this example, of course!

Getting started

The first step is to make sure that you have a recent version of Xcode (at the time of writing, Xcode 11) installed on your Mac. If you're using earlier versions of Xcode, this code may not compile, and you may not be able to follow directions. Xcode is free and may be downloaded from the Mac App Store or from Apple's developer portal.

We'll also be using the Swift programming language, instead of Apple's older programming language for iOS, Objective-C. Almost all of this book would be directly applicable to an Objective-C application. The underlying application programming interfaces (APIs) used in iOS are generally the same.

The Swift language has been evolving since its first release. This book uses Swift 5, which is supported in Xcode 10 and above.

Go ahead and open up Xcode, and create a new application. We'll be creating a new Single View Application (Figure 1-1).

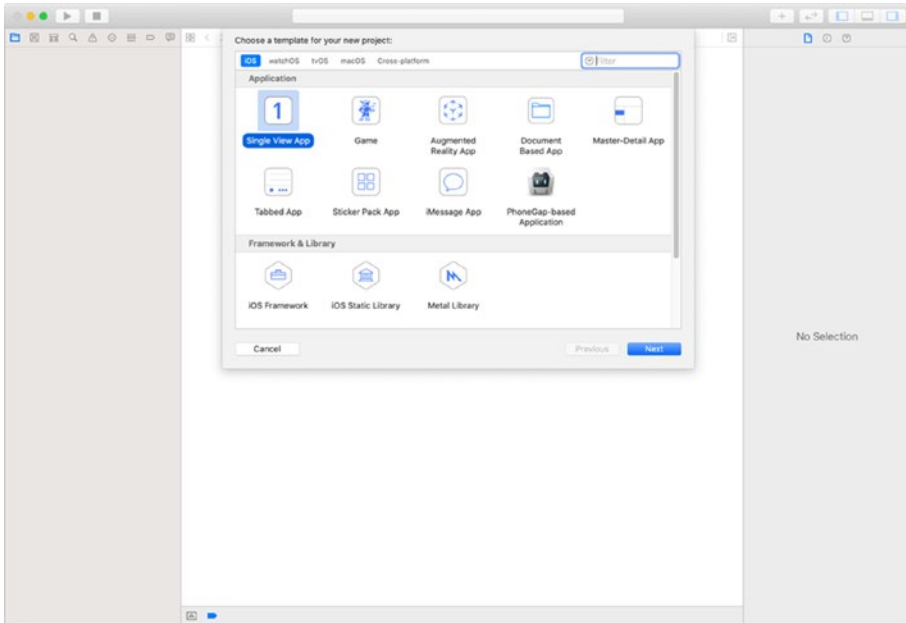


Figure 1-1. *New project window in Xcode*

Click the Next button, and then name your new project on the options dialog, as seen in Figure 1-2. I'm going to call the new application FirstMapsApp and give it an organization identifier of com.buildingmobileapps.maps and use my name for the Organization Name.

Be sure to choose Swift as the Language.

For this project, we will not be using SwiftUI – we will be using UIKit as the application framework. Leave the SwiftUI check box unchecked.

We do not need to include Core Data in our project – Core Data is an Apple technology used for storing data locally on iOS, and we won't need it for this example. We won't be using Core Data in this book.

You can also uncheck Include Unit Tests and Include UI Tests, as we won't be setting up any tests for this project.

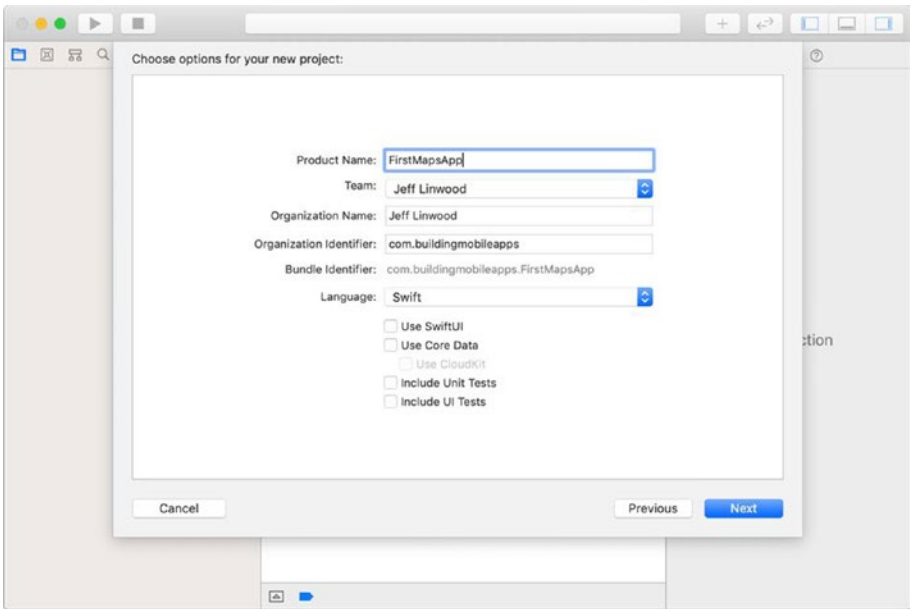


Figure 1-2. *New project options for an iOS app*

Click Next, and save the project in a convenient location. You can create a Git repository for your code if you want, but we won't be directly addressing source control in this book. It's always a good idea to keep up with Git commits as your project goes along, so that you can easily roll back to a working copy.

After saving your project, Xcode will open your project and present an overview of your application (Figure 1-3).

CHAPTER 1 CREATING YOUR FIRST MAPKIT APP

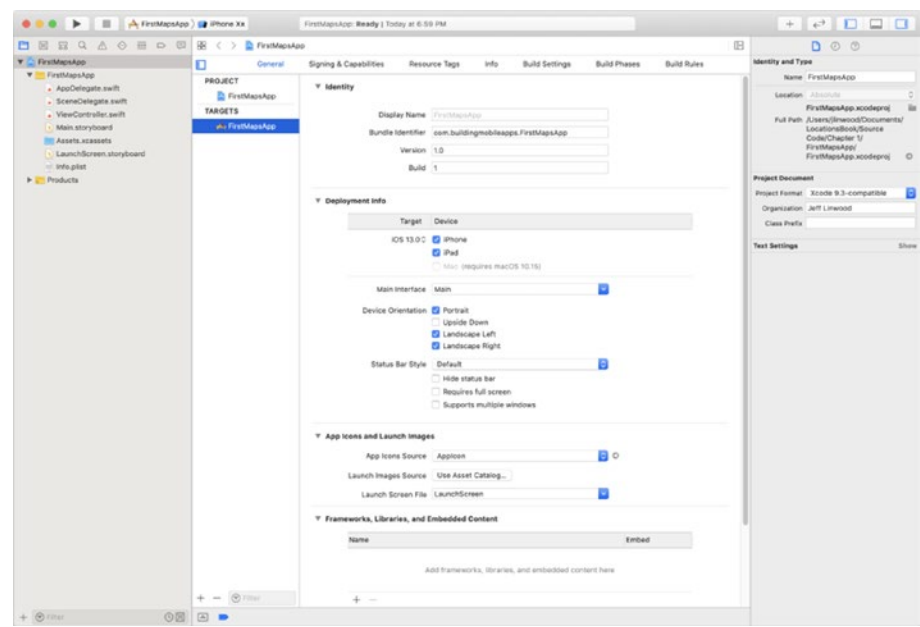


Figure 1-3. Project overview

You should now have a working Xcode project – go ahead and run it in one of the iOS Simulators, for instance, the iPhone XR (Figure 1-4).

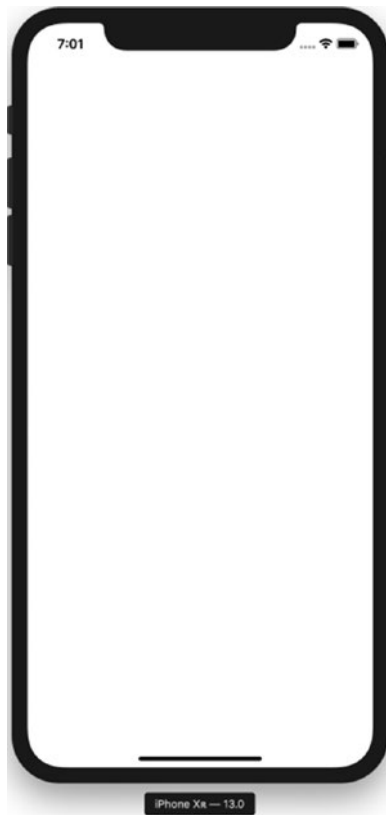


Figure 1-4. *New iOS application running in a simulator*

You should expect to see a blank screen, as we have not written any code for our application yet. If you do, your development environment is set up and ready to go for the rest of this chapter.

Adding a map

Now it's time to add a map to our view controller. Select the storyboard on the left-hand side; it is the file named `Main.storyboard`. Once the storyboard opens, select the View Controller Scene.

CHAPTER 1 CREATING YOUR FIRST MAPKIT APP

In the upper right-hand corner of your Xcode window, choose the left-most button (the Object library), which is the button with the plus sign in the previous figure, as shown in Figure 1-5.

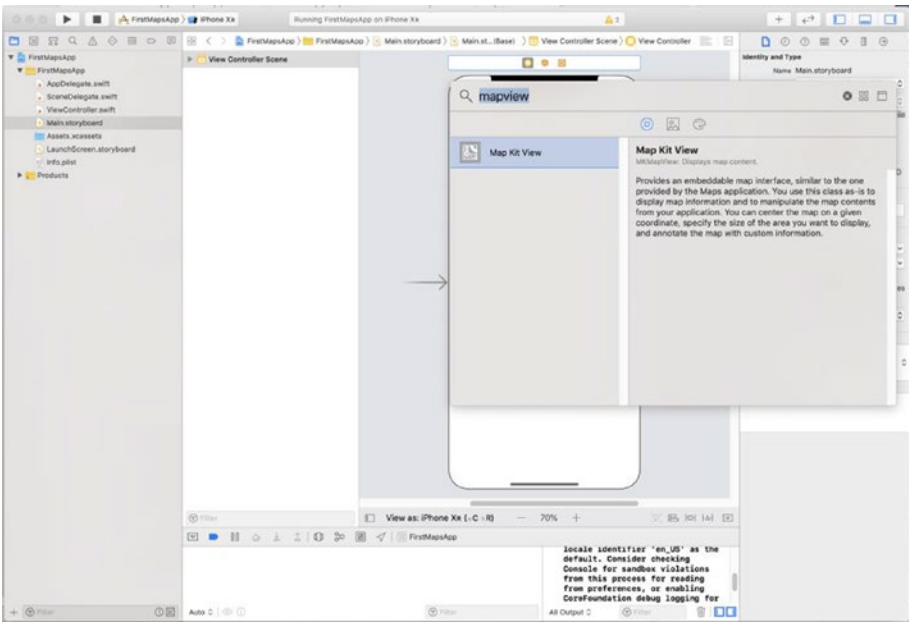


Figure 1-5. Choosing an *MKMapView* map from the Object library in Xcode

Either type Map into the search box underneath the list or scroll down until you find the Map Kit View. Once you have found the Map Kit View, drag it onto your view controller (Figure 1-6).

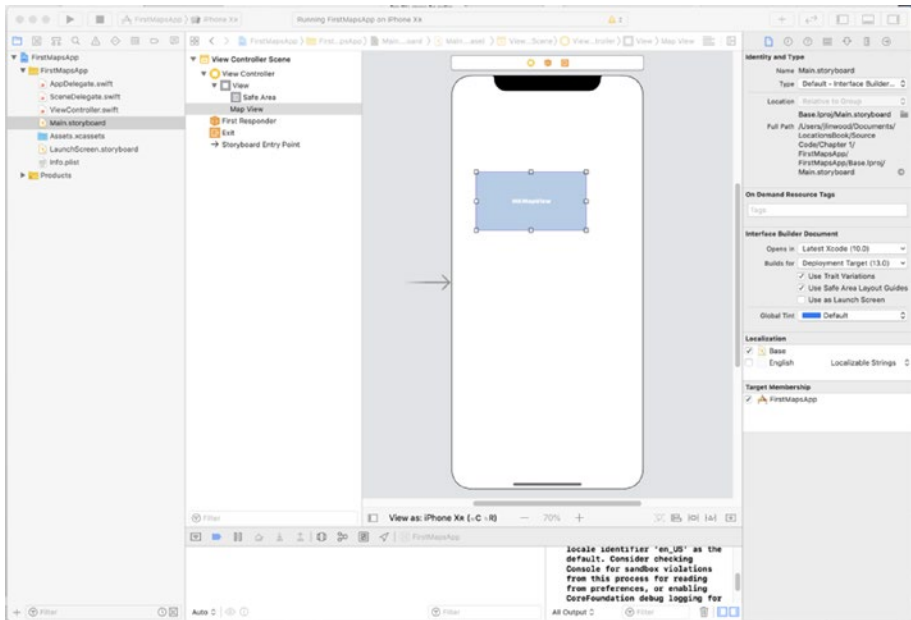


Figure 1-6. Map View on storyboard

The map view won't automatically expand to fill the whole screen, so you will need to do that yourself by dragging the edges of the map view to fill the extent of the view. In Figure 1-7, you can see how the map view fills the entire view controller on an iPhone XR device with a notch at the top.

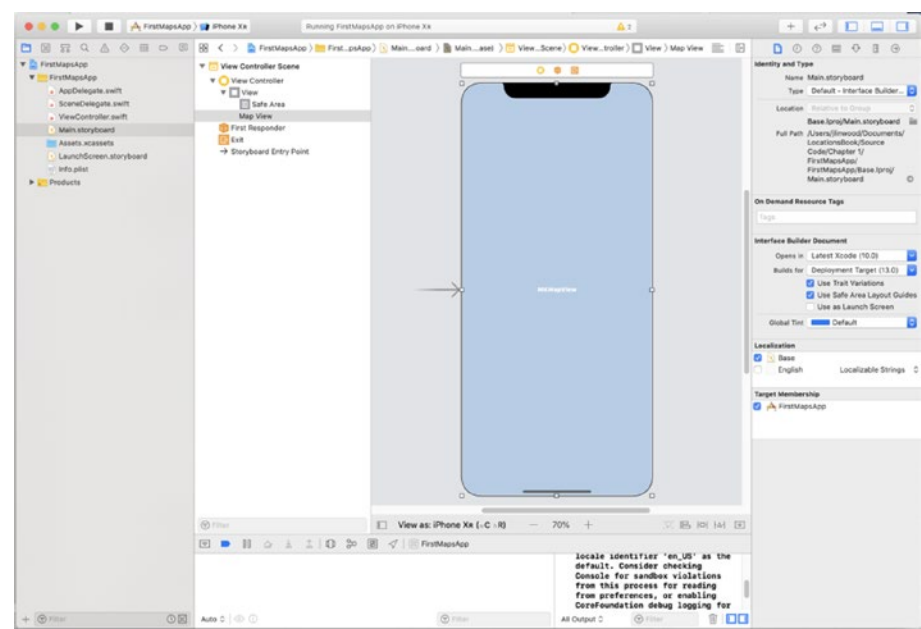


Figure 1-7. Map View fills view

Even though we dragged the edges of the map view out to the edges of the view controller’s view, that doesn’t mean that the map view will use the entire screen on all sizes of the iPhone and iPad. To make the map view fill the view controller’s view (also known as its parent view), we will need to add constraints to the map view.

On the right-hand side of the toolbar underneath your view controller, you will see five icons – the first icon is usually grayed out. The third icon (Add New Constraints) opens up the Add New Constraints dialog box, which we can use for our layouts.

Uncheck the “Constrain to margins” check box, as we are going to fill the entire view with the map, not leaving any margins. Go ahead and select the faint dashed red line for all four constraints (top, bottom, left, and right). After selecting them, make sure that all of the values are 0, and press the “Add 4 Constraints” button (Figure 1-8).

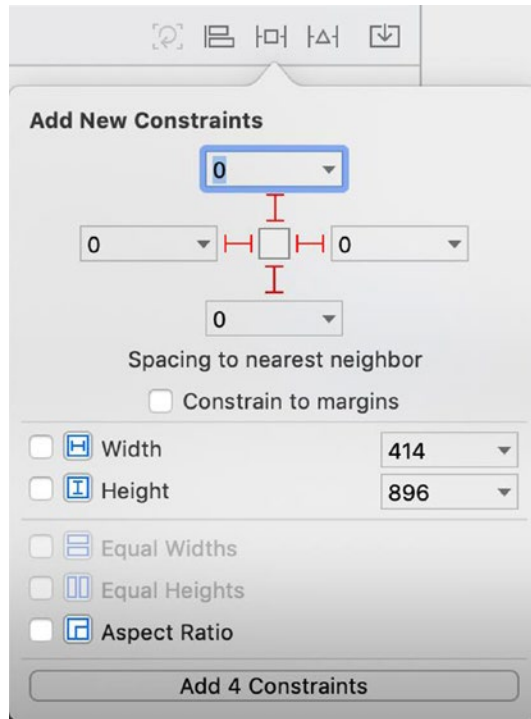


Figure 1-8. *Adding constraints to a Map View*

Your map view will now properly fill up the entire screen on an iPhone or iPad. If you would like to double-check this, select the Map View on the storyboard. Next, choose the fifth icon on the right-hand side, the Size Inspector, and you will see that you have constraints for all four sides of your Map View.

Now try running your iOS app, and you will see that you have a nice, large map on your app – as seen in Figure 1-9.

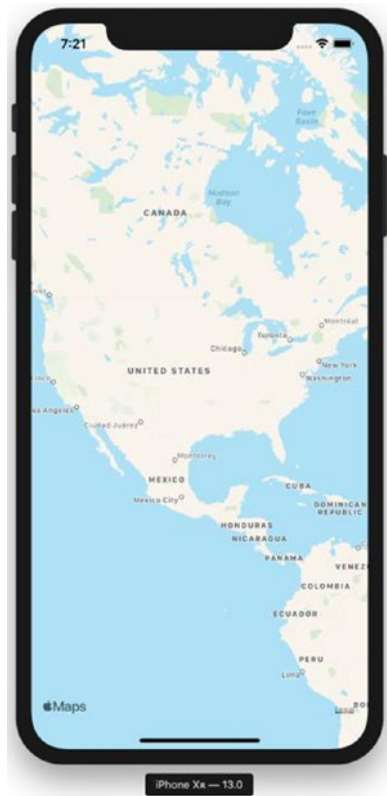


Figure 1-9. *An iOS app with a full-screen map view*

This was a pretty straightforward process to get the map up and running and didn't even involve writing any code in Swift.

Adding a pin to your map

Now that we have our map, it's time to add a pin that shows our home city!

Before we add the pin to the map, we will need to create an outlet for the map, named `mapView`, using Xcode's Assistant. While you have the `Main.storyboard` editor open, choose the **Assistant** view from the **Editor** menu. You'll see the `ViewController` class open up next to your view controller in the storyboard (Figure 1-10).

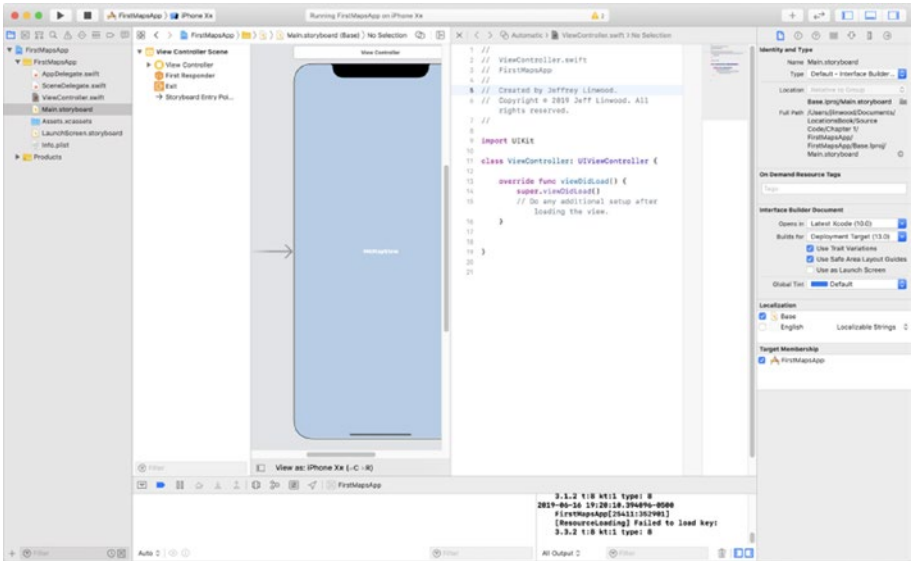


Figure 1-10. Xcode Editor and Assistant view

Select the map view on the storyboard or on the outline view, hold down the Control key, and then drag an outlet into the ViewController class, as shown in Figure 1-11.

CHAPTER 1 CREATING YOUR FIRST MAPKIT APP

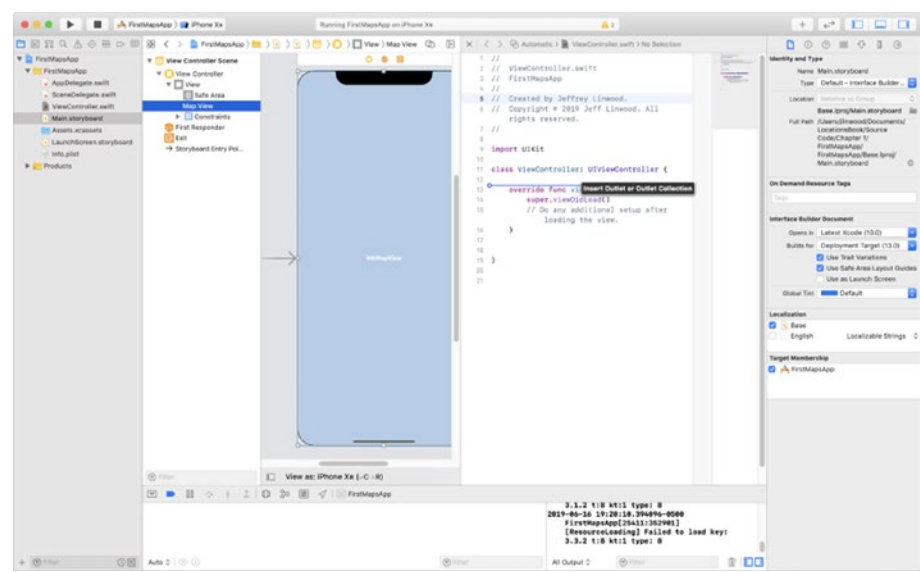


Figure 1-11. Creating an outlet

After creating the outlet, name the outlet `mapView` in the dialog box that appears (Figure 1-12).

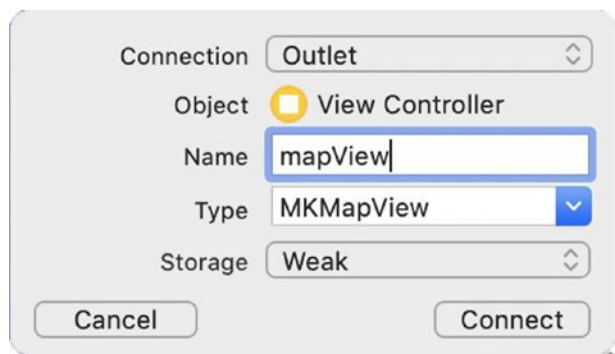


Figure 1-12. Naming the outlet

You'll notice that the `ViewController` class will no longer compile – that is because our map is an `MKMapView`, part of the MapKit framework. We need to import this framework into our `ViewController` class so that we

can use classes from the MapKit framework. Otherwise, Xcode will show errors when we try and build our project.

Add the following line right below the `import UIKit` statement to import the MapKit framework.

```
import MapKit
```

Beyond maps themselves, the MapKit framework has a wide range of functionality. With MapKit, we represent locations on the map as annotations. Annotations implement the `MKAnnotation` protocol, which consists of a latitude and longitude coordinate pair and an optional title and subtitle. The MapKit framework comes with a basic implementation of `MKAnnotation`, the `MKPointAnnotation` class, but for most apps, you will want to create your own implementation of `MKAnnotation`. In this chapter, we will use `MKPointAnnotation`, but the later chapters of this book will use our own implementation, so you can see how it works both ways.

Once you have an annotation (or many annotations), you can just add it to the map using the `addAnnotation()` or `addAnnotations()` method on the `MKMapView` class.

Annotations are not the actual pin that the map displays – those are annotation views, which are subclasses of the `MKAnnotationView` class. By default, you will get an `MKPinAnnotationView`, which is the standard red pin that you see in many mapping apps. You can customize the pin color a little, but for most apps, you will want to put in your custom images. We'll use our own custom images in the next chapters of this book.

To create an annotation as an `MKPointAnnotation`, we do need to be able to create a coordinate, which we can do with `CLLocationCoordinate2DMake()`. For our purposes in this chapter, we are going to add all of the code to the `viewDidLoad()` method in the `ViewController` class. Xcode created this method for you when you generated a new project.

This method is currently empty except for a call to `super.viewDidLoad()`. Leave that line of code in the `viewDidLoad()` method, and place this code beneath that.

Pass in the latitude and the longitude (as double values) to create the coordinate. The `MKPointAnnotation` will need this coordinate set, such as in the following code:

```
let austin = MKPointAnnotation()
austin.coordinate = CLLocationCoordinate2DMake(30.25, -97.75)
```

The longitude for Austin is going to be negative, because Austin, Texas, is in the Western Hemisphere. The latitude is positive because the city is in the Northern Hemisphere.

To give the annotation a title, we can simply set the title property

```
austin.title = "Austin"
```

And then after setting the title and coordinate properties, we can call one method on the map view to add the annotation

```
mapView.addAnnotation(austin)
```

Run this class (Listing 1-1), and you will see the iPhone app displaying the Austin pin, after you scroll the map to show Texas on your Simulator. Go ahead and change the pin to your city or to any other location you want. Add more pins for different locations!

Listing 1-1. The `ViewController` class with a pin that displays on the Map view

```
import UIKit
import MapKit

class ViewController: UIViewController {

    @IBOutlet weak var mapView: MKMapView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
```

```
let austin = MKPointAnnotation()  
austin.coordinate = CLLocationCoordinate2DMake(30.25,  
-97.75)  
austin.title = "Austin"  
mapView.addAnnotation(austin)  
}  
}
```

Figure 1-13 shows the iPhone app with the Austin pin.

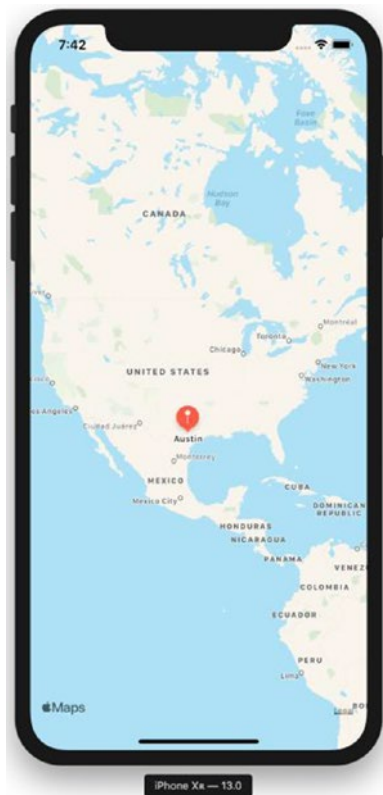


Figure 1-13. The completed iOS app, showing the pin for Austin on the map