EE 443/ EE 593 Mobile Application Development

George Torres

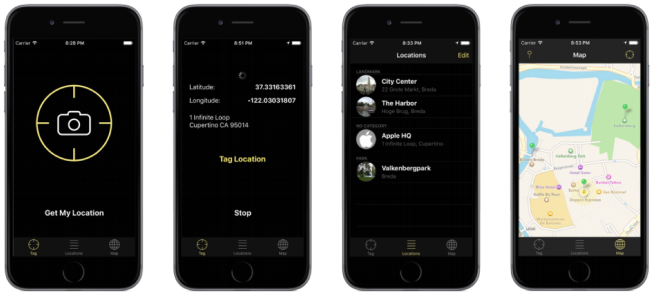
Lecture 16: Chapter 22: Get Location Data

George Torres

MyLocations

Starting a new app named MyLocations with the following functionality.

- Obtain GPS coordinates and the corresponding street address with your iPhone by pressing the Get My Location button to obtain GPS coordinates and the corresponding street address.
- Keeps a list of spots you find interesting.
- Save locations along with a description and a photo in your list

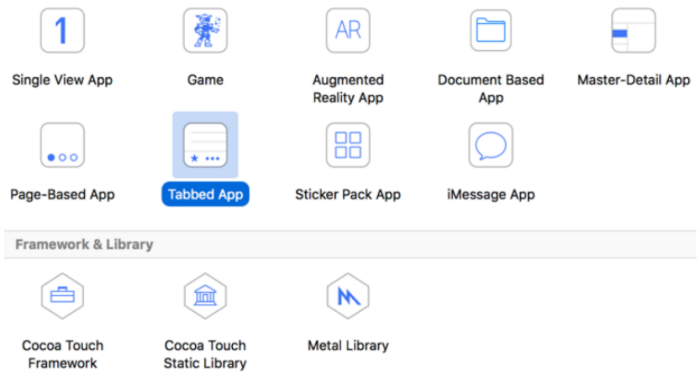


Objectives for chapter

- Create a tab bar-based app with two tabs.
- Get GPS Coordinates by learning to use the CoreLocation framework to get the user's current location.
- Display coordinates by setting up the UI for the first tab and displaying location information on screen.

Create project

- Create a new project, choosing the tabbed app template



- Name the product MyLocations

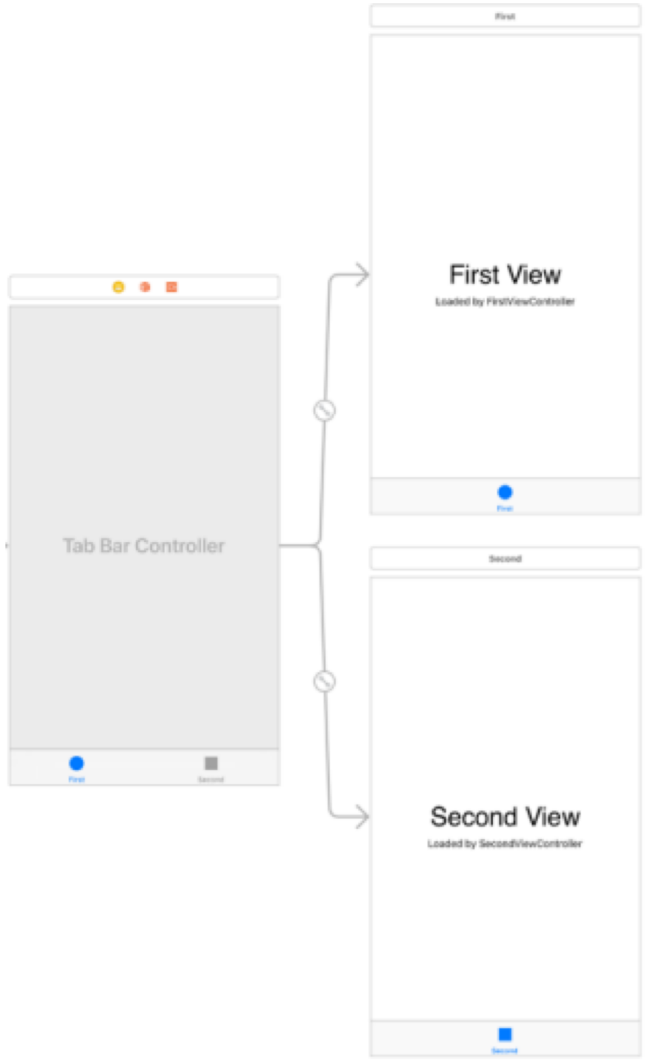
Tabbed App template

The app starts with three view controllers

- The root controller: a UITabBarController that contains the tab bar and performs the switching between the different screens.
- A view controller for the First tab.
- A view controller for the Second tab.

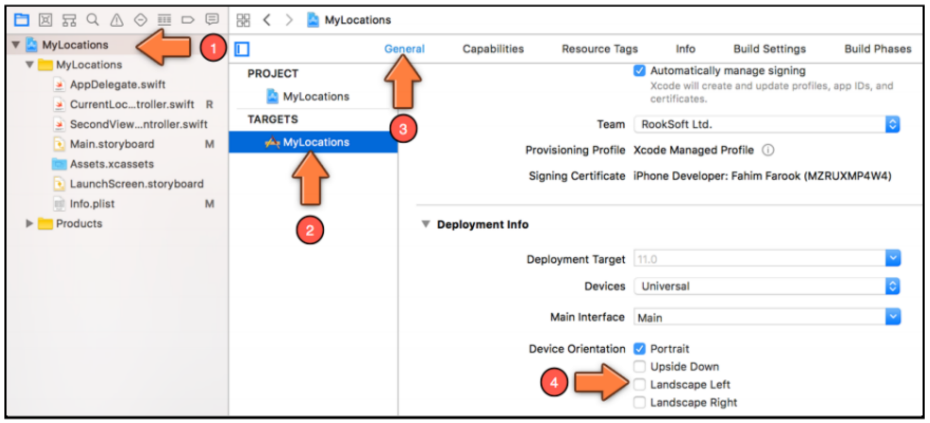
Both tabs have their own view controllers with the default names FirstViewController and SecondViewController. Today we'll only be working with the First tab and it’s view controller.

Starting Storyboard

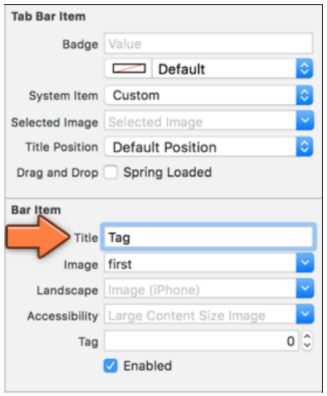


First Tab

- First we'll rename view controller to CurrentLocationViewController
- In project settings we'll enable only portrait orientation.



- Select the Tab Bar Item object from the First view, go to the Attributes inspector and change the Title to Tag.

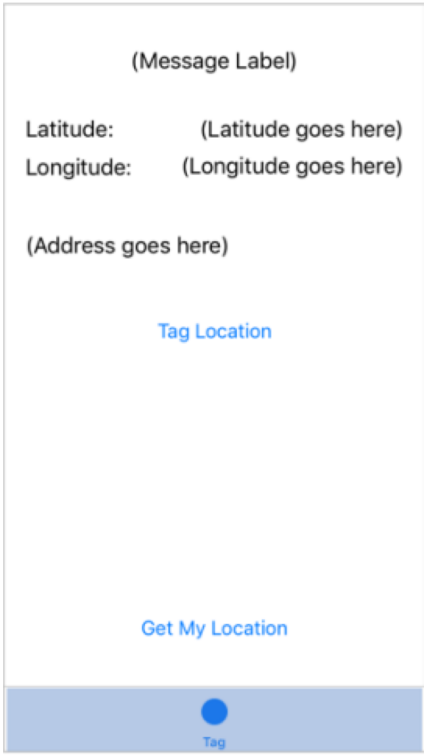


First tab UI

- In the CurrentLocationViewController add the following

```
@IBOutlet weak var messageLabel: UILabel!
@IBOutlet weak var latitudeLabel: UILabel!
@IBOutlet weak var longitudeLabel: UILabel!
@IBOutlet weak var addressLabel: UILabel!
@IBOutlet weak var tagButton: UIButton!
@IBOutlet weak var getButton: UIButton!
@IBAction func getLocation() {
  // do nothing yet
}
```

First tab UI design



Core Location

- Gives you a way to know exactly where you are on the globe
- Can supply our app with the user's current latitude and longitude.
- Uses communication with GPS satellites, or Wi-Fi and cell tower triangulation to get location automatically.
  - Cell tower triangulation and Wi-Fi positioning: fast but inaccurate.
  - GPS: accurate but slow.
  - Core Location turns the location readings from its various sources into a useful number without any extra work. Sending the location data to the app as soon as it gets it, following up with more and more accurate readings

Setting up Core Location

- We first import the Core Location framework
- Core Location works via delegate, so we conform the view controller to the CLLocationManagerDelegate protocol by adding the following.

```
class CurrentLocationViewController: UIViewController,
                                     CLLocationManagerDelegate {
```

```
let locationManager = CLLocationManager()
```

```
func locationManager(_ manager: CLLocationManager,
        didFailWithError error: Error) {
  print("didFailWithError \(error)")
}

func locationManager(_ manager: CLLocationManager,
  didUpdateLocations locations: [CLLocation]) {
  let newLocation = locations.last!
  print("didUpdateLocations \(newLocation)")
}
```

For now we simply print to the console.

Getting current location

Add the following code and connect the Get My Location button to getLocation():

```
@IBAction func getLocation() {
  locationManager.delegate = self
  locationManager.desiredAccuracy =
                  kCLLocationAccuracyNearestTenMeters
  locationManager.startUpdatingLocation()
}
```

- Set the view controller as the location manager's delegate
- Set the accuracy of received locations to ten meters.
- Start the location manager
  - Once called the view controller will continuously receive GPS coordinates until stopped. This wastes power and we will change this later.

Asking for permission

- Add the following lines to the top of getLocation():

```
let authStatus = CLLocationManager.authorizationStatus()

if authStatus == .notDetermined {
  locationManager.requestWhenInUseAuthorization()
  return
}
```

- Checks the current authorization status. If it's .notDetermined, the app will request “When In Use” authorization. Allowing the app to get location updates while open.
  - There is also “Always” authorization, allowing the app to check the user’s location even when it is not active.

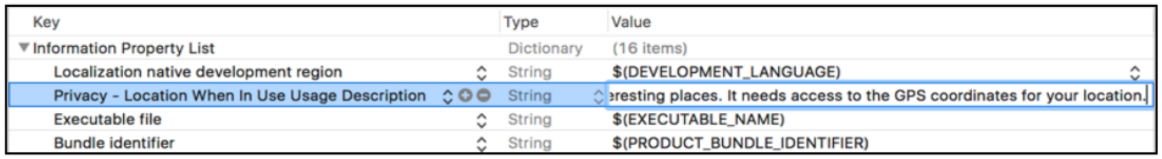
Adding to Info.plist

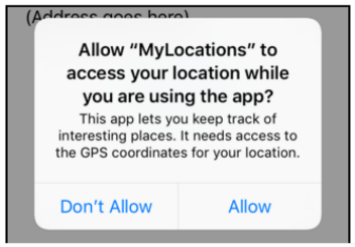
We must also add a special key to the app's Info.plist.

Add a new row to Info.plist file.

- For the key, type "NSLocationWhenInUseUsageDescription".
- Type the following text in the Value column

  "This app lets you keep track of interesting places. It needs access to the GPS coordinates for your location. This description tells the user what the app wants to use the location data for."





App now asks for permission

Permission errors

- If we deny permission, the app will never get the user's location. Error message appears in debug.
- To handle this in a user friendly way we add the following method:

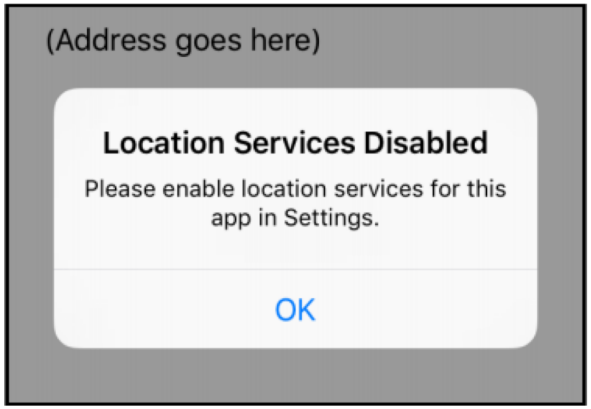
```
func showLocationServicesDeniedAlert() {
  let alert = UIAlertController(
    title: "Location Services Disabled",
    message: "Please enable location services for this app in Settings.",
    preferredStyle: .alert)

  let okAction = UIAlertAction(title: "OK", style: .default,
                               handler: nil)
  alert.addAction(okAction)

  present(alert, animated: true, completion: nil)
}
```

- This pops up an alert informing the user that this app needs location services to be enabled.

Show alert

- To show this alert, add the following lines to getLocation(), just before you set the locationManager's delegate:

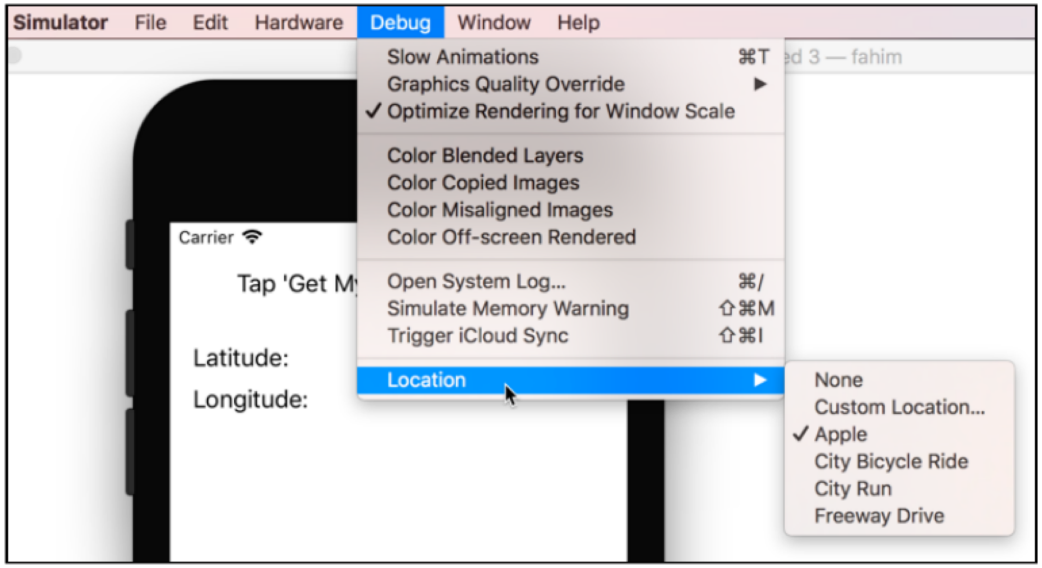
```
if authStatus == .denied || authStatus == .restricted {
  showLocationServicesDeniedAlert()
  return
}
```



The alert now appears informing the user to enable location services for your app. Which they can do from the iPhone's Settings app.

Unknown location

- If we run the app on the simulator we get an error, code '0' meaning location unknown.
- To fix this we must simulate the location



While the app is running, from the Simulator's menu bar at the top of the screen, choose Debug → Location → Apple

Asynchronous operations

- Obtaining a location is an example of an asynchronous process.
- Sometimes after you start an operation, you have to wait until it gives you the results.
  - Core Location, can take a couple of seconds to get the first location reading and more for a location accurate enough to use.
- If an asynchronous operation starts, the app will continue on without waiting for the results. The asynchronous process is said to be operating “in the background”. As soon as the operation is done, the app is notified through a delegate so that it can process the results.

Synchronous operations

- The opposite is synchronous. If a synchronous operation starts, the app won't continue until that operation is done.
  - If we did that with CLLocationManager the app would be totally unresponsive for the couple of seconds that it takes to get a location fix. Leading to a bad experience for the user.
- Operations that take longer than a fraction of a second should be performed in an asynchronous manner.
- iOS has something called the "watchdog timer". If your app is unresponsive for too long, then under certain circumstances, the watchdog timer will kill your app.

Getting coordinates

- Add the following instance variable to the class:

```
var location: CLLocation?
```

  - We store the user's current location in this variable
- Add code to the end of locationManager(_:didUpdateLocations:):

```
    location = newLocation    // Add this
    updateLabels()            // Add this
}
```

  - The locationManager(_:didUpdateLocations:) delegate method gives an array of CLLocation objects with the user's current coordinates. We set location to the last CLLocation object from the array, being the most recent object, and call updateLabels (a new method) to display its coordinates in the corresponding labels.

Display coordinates

- Add the updateLabels() method:

```
func updateLabels() {
  if let location = location {
    latitudeLabel.text = String(format: "%.8f",
                                location.coordinate.latitude)
    longitudeLabel.text = String(format: "%.8f",
                                 location.coordinate.longitude)
    tagButton.isHidden = false
    messageLabel.text = ""
  } else {
    latitudeLabel.text = ""
    longitudeLabel.text = ""
    addressLabel.text = ""
    tagButton.isHidden = true
    messageLabel.text = "Tap 'Get My Location' to Start"
  }
}
```

  - If there is a valid location object, we convert the latitude and longitude, from type Double to type String and then put them into the labels. We show the tag button & empty the messageLabel
  - If not all labels are set to empty except for messageLabel, and the tag button is hidden.

Format strings

- We use string interpolation to put values into strings.
- We've done this before so why not simply do the following?

```
latitudeLabel.text = "\(location.coordinate.latitude)"
```

- No control over how the value appears. We want the value to be shown with 8 digits behind the decimal point. To do that we need to use a format string
- A format string uses placeholders that will be replaced by the actual value during runtime.

```
String(format: "%.8f", location.coordinate.latitude)
```

- The %.8f format specifier does the same thing as %f: it takes a decimal number and puts it in the string. The .8 means that there should always be 8 digits behind the decimal point.