Lab 01: Maps on iOS using MapKit

Prerequisites

You will need a development environment, either a Mac or Windows PC remotely connected to a Mac with XCode and Xamarin.iOS tools installed. We will be using the iOS simulator to test the code we are building, so make sure you have successfully built and deployed a test project first. See the **Xamarin.iOS** setup documentation if you need help getting your environment setup:

http://docs.xamarin.com/guides/ios/getting_started/installation/

Assets

This exercise uses a set of starter projects which are located at:

Content/Exercise/MapsiOS/Lab Resources

Lab Goals

The goal of this lab will be to demonstrate how to use MapKit on iOS. We will cover how to show a map, change the map style, interact with the map, plot points on a map, and how to query for nearby locations and addresses.

The code samples are structured to incrementally add functionality as you progress through the chapter. There are five sample projects included in the solution:

- **MappingIOSLab_demo1** – Show initial map, change map style, enable/disable interactions, and show user location.
- **MappingIOSLab_demo2** – Add a point annotation, customize the annotation and add accessory views.
- **MappingIOSLab_demo3** – Draw custom points on the map with reusable views.
- **MappingIOSLab_demo4** – Draw circles, polygons, and other shapes.
- **MappingIOSLab_demo5** – Lookup a street address, and lookup locations in a region.

The code introduced in this chapter is already present in the demo projects, but it's commented out.

The tasks for this lab are marked with a `// TODO:` task identifier and can be found quickly using the **Tasks** pane in Xamarin Studio or Visual Studio without having to navigate to individual files.

Open the Starting Solution

1. Launch Xamarin Studio and open the starting solution – MappingIOSLab_Begin**MappingIOSLab.sln**.

MappingIOSLab_demo1 – Add a basic map

1. Set the MappingIOSLab_demo1 project as the startup project (right click, choose 'Set as Startup Project').
2. Run the application in the iPhone simulator or on a physical device – you should just see a white screen. Go ahead and stop the app.
3. The first thing we will do is add a new `MKMapView` to our view. This requires using the `MonoTouch.MapKit` namespace. Locate the comment `TODO: Step 1a - add the map view` and uncomment the following lines in the `ViewDidLoad()` method in MappingAppViewController.cs:

```
var map = new MKMapView(UIScreen.MainScreen.Bounds);
View.Add(map);
```

4. Run the application again – you should now see a basic map view.
5. Next, we will change the map style to show the satellite view. Locate the comment `TODO: Step 1b - change the map style` and try uncommenting each of the commented lines to see how it changes the display.

```
// map.MapType = MKMapType.Standard;
map.MapType = MKMapType.Satellite;
// map.MapType = MKMapType.Hybrid;
```

6. Run the application again – you should now see a satellite map view.
7. Now we can see how to lock the map position by preventing the user from zooming the map view, and preventing them from scrolling the window. This can be useful if you want to set the map view to show a specific area and prevent interactions. Locate the comment `TODO: Step 1c - enable/disable interactions` and uncomment the following lines:

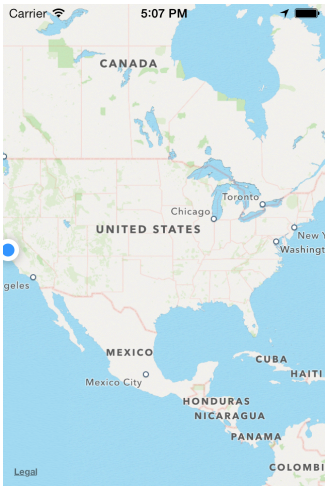
```
map.ZoomEnabled = false;
map.ScrollEnabled = false;
```

8. Finally, you can show the user's location on the map just by setting **ShowsUserLocation** to true. On iOS8 and beyond, you also have to request permission from the location manager and include the **NSLocationAlwaysUsageDescription** or **NSLocationWhenInUseUsageDescription** permission in your **info.plist**. The permission and code has already been added for you.
9. Locate the comment `TODO: Step 1d - show user location` and uncomment the following lines:

```
if (UIDevice.CurrentDevice.CheckSystemVersion (8, 0)) {
    locationManager.RequestAlwaysAuthorization ();
}
```

```
map.ShowsUserLocation = true;
```

10. Run the application one last time to see the map no longer allowing interactions, and a pulsing blue circle on the left.



MappingIOSLab_demo2 – Working with annotations

1. Set the MappingIOSLab_demo2 project as the startup project (right click, choose 'Set as Startup Project'). You can run the project if you'd like – it should be just as left off after demo1.
2. The first thing we want to see is how to add a point onto the map. This is done by adding an annotation onto the map – specifically a `MKPointAnnotation`. Locate the comment `TODO: Step 2a - Add a point annotation` in `ViewDidLoad()` and uncomment the following lines:

```
  map.AddAnnotation(new MKPointAnnotation()
{
    Title = "MyAnnotation",
    Coordinate = new MonoTouch.CoreLocation.CLLocationCoordinate2D(42.
364260, -71.120824)
});
```

3. If you run the app now, you'll notice a new point drop-pin has been added to the northeast of New York on our map. If you tap the pin, you will see the Title we provided "MyAnnotation" – here is where you could display more data about this point.
4. We can customize both the pin color itself as well as the callout that appears when the pin is tapped by creating a custom delegate that returns a custom `MKPinAnnotationView`. Locate the comment `TODO: Step 2b - Customize annotation view via GetViewForAnnotation delegate` in MappingAppViewController.cs and uncomment the following code:

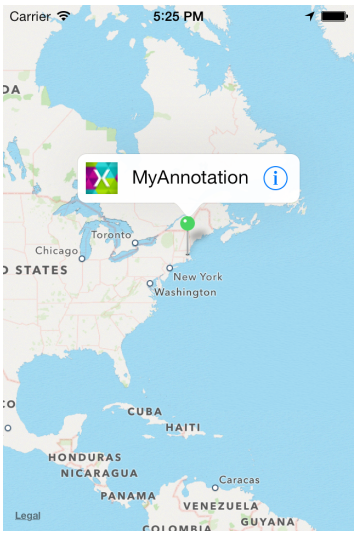
```
map.GetViewForAnnotation = delegate(MKMapView mapView, NSObject annota
tion)
```

```
{
    if (annotation is MKUserLocation)
        return null;

    MKPinAnnotationView pinView = (MKPinAnnotationView)mapView.Dequeue
ReusableAnnotation(pId);
    if (pinView == null)
    {
        pinView = new MKPinAnnotationView(annotation, pId);

        // TODO: 2e: Move one-time setup to here
    }

    pinView.PinColor = MKPinAnnotationColor.Green;
    pinView.CanShowCallout = true;

    // TODO: Step 2d: Add accessory views to the pin
    pinView.RightCalloutAccessoryView = UIButton.FromType(UIButtonType
.DetailDisclosure);
    pinView.LeftCalloutAccessoryView = new UIImageView(UIImage.FromFil
e("icon-29.png"));

    return pinView;
};
```

If you have used a `TableViewController` or `CollectionViewController`, then you should be familiar with the `DequeueReusableAnnotation` call above – used for virtualizing and re-using pin views.

The code above sets the pin color to green, and uses `CanShowCallout` to ensure it can be tapped – setting this to false would prevent that interaction.

In step 2d above you can see how we can customize the accessory view that pops up when you tap on a pin. You can see what we are using a built in button type on the right (`UIButtonType.DetailDiscosure`), and a custom icon on the left.



Assuming that the color of your pin is always the same, and not changing per pin, you can move those calls to the Step 2e comment so that they aren't set every time a pin view is needed. If you make this change, your code will look something like the following:

```
map.GetViewForAnnotation = delegate(MKMapView mapView, NSObject annota
tion)
{
    if (annotation is MKUserLocation)
        return null;

    MKPinAnnotationView pinView =
        (MKPinAnnotationView)mapView.DequeueReusableAnnotation(pId);

    if (pinView == null)
    {
        pinView = new MKPinAnnotationView(annotation, pId);

        // One-time setup
        pinView.PinColor = MKPinAnnotationColor.Green;
        pinView.CanShowCallout = true;

        // Add accessory views to the pin
        pinView.RightCalloutAccessoryView =
            UIButton.FromType(UIButtonType.DetailDisclosure);
        pinView.LeftCalloutAccessoryView =
            new UIImageView(UIImage.FromFile("icon-29.png"));
    }

    return pinView;
};
```

MappingIOSLab_demo3 – Drawing custom points

1. Set the MappingIOSLab_demo3 project as the startup project. You can run the project if you'd like – it should be just as left off after demo2.
2. Locate the comment `TODO: Step 3a - remove old …` and delete the old setting of `map.GetViewForAnnotation` delegate in `MappingAppViewController.ViewDidLoad`:

```
map.GetViewForAnnotation = delegate(MKMapView mapView, NSObject annota
tion)
{

…
}
```

Similar to how we customized the pins on our map by providing a custom delegate in-line in the previous demo, we will now explicitly create a delegate class so that we can do even more customizations.

3. Create a new `MyMapDelegate` class using `MKMapViewDelegate` as the base class. Locate the comment `TODO: Step 3b – Setup a map delegate …` and uncomment the entire implementation:

```
class MyMapDelegate : MKMapViewDelegate
{
   …
}
```

4. Next we will override `GetViewForAnnotation()` as part of our new `MyMapDelegate` similar to before to allow for custom annotation views. We could have just moved what we had before in our `ViewDidLoad()`, but instead we will update it to support different annotation types.
5. As part of that, we will also need to override `ChangedDragState()` which is called when our annotation state has been changed. Here we will want to change the drag state to `None`.
6. Locate the comment `TODO: Step 3c – Review new …` and uncomment the following method implementations:

```
public override MKAnnotationView GetViewForAnnotation (MKMapView mapVi
ew, NSObject annotation)
{
    MKAnnotationView anView;

    if (annotation is MKUserLocation)
        return null;

    if (annotation is CustomAnnotation) {

        // show custom annotation
        anView = mapView.DequeueReusableAnnotation (cId);

        if (anView == null)
            anView = new MKAnnotationView (annotation, cId);

        anView.Image = UIImage.FromFile ("icon-29.png");
        anView.CanShowCallout = true;
        anView.Draggable = true;
        anView.RightCalloutAccessoryView = UIButton.FromType (UIButton
Type.DetailDisclosure);

    } else {

        // show pin annotation
        anView = (MKPinAnnotationView)mapView.DequeueReusableAnnotatio
n (pId);

        if (anView == null)
            anView = new MKPinAnnotationView (annotation, pId);

        ((MKPinAnnotationView)anView).PinColor = MKPinAnnotationColor.
Green;
```

```
        anView.CanShowCallout = true;
    }

    return anView;
}
// Because we set anView.Draggable = true; we need to also
// handle the Ending drag event, or else our annotation will
// forever be "floating" over the map
public override void ChangedDragState (MKMapView mapView,
     MKAnnotationView annotationView,
     MKAnnotationViewDragState newState,
     MKAnnotationViewDragState oldState)
{
    switch (newState) {
       case MKAnnotationViewDragState.Ending:
           annotationView.SetDragState(
                  MKAnnotationViewDragState.None, false);
       break;
    }
}
```

7. Set the map's `Delegate` property in `ViewDidLoad()` to the new `MyMapDelegate` class. Locate the comment `TODO: Step 3d – specify a custom map delegate` and uncomment the following lines:

```
var mapDelegate = new MyMapDelegate();
map.Delegate = mapDelegate;
```

8. You can see that we now are checking the annotation to see if it is a custom subclassed annotation, and change what the view looks like that we return. This is how we can support different pins and items shown on a map.
9. Now let's add the new `CustomAnnotation` class – note that it derives from `MKAnnotation`. Locate the comment `TODO: Step 3e – Create custom annotation type` and uncomment the following:

```
public class CustomAnnotation : MKAnnotation
{
    private string _title;
    public override string Title { get { return _title; } }

    public override CLLocationCoordinate2D Coordinate { get; set; }

    public CustomAnnotation (string title, CLLocationCoordinate2D coor
d)
    {
        _title = title;
        this.Coordinate = coord;
    }
}
```

10. We now need to add our custom annotation. Locate the comment `TODO: Step 3f – add a custom annotation` and uncomment the following lines:

```
map.AddAnnotation(new CustomAnnotation("CustomSpot", new MonoTouch.Cor
eLocation.CLLocationCoordinate2D(38.364260, -68.120824)));
```

11. Run the app now to see our custom annotation pin.
12. Notice that the code in `MyMapDelegate.GetViewForAnnotation()` is using custom IDs for the call to `DequeueReusableAnnotation()` and when creating instances of `MKAnnocationView` and `MKPinAnnotationView` – this is how the map virtualizes items to handle showing large numbers of items.
13. Now let's add event handling to when the user taps on an accessory view callout from a pin. Locate the comment `TODO: Step 3g - Add event handler …` and uncomment the following lines:

```
public override void CalloutAccessoryControlTapped (MKMapView mapView,
 MKAnnotationView view, UIControl control)
{
    var customAn = view.Annotation as CustomAnnotation;

    var alert = new UIAlertView(customAn != null ? "Custom Annotation"
 : "Generic Annotation",
        customAn != null ? customAn.Title : "Title Text",
        null, "OK");

    alert.Show ();
}
```

14. We can also add event handling to when the pin marker is tapped. Locate the comment `TODO: Step 3h - Add event handler …` and uncomment the following lines:

```
public override void DidSelectAnnotationView(MKMapView mapView, MKAnno
tationView view)
{
    var alert = new UIAlertView("Marker was tapped", "Marker tapped",
null, "OK");
    alert.Show ();
}
```

15. Run the app one last time to see how your event handlers are fired.

MappingIOSLab_demo4 – Drawing custom points

1. Set the MappingIOSLab_demo4 project as the startup project.

   Now we will see how to draw custom overlays on our maps. This could be useful for illustrating regions of data, or drawing our custom overlays.

2. First, let's draw a circle overlay. We do this by adding a circle overlay to our map and specifying its position and size in meters. Locate the comment `TODO: Step 4a - draw a circle overlay` and uncomment the following lines:

```
var circleOverlay = MKCircle.Circle(new CLLocationCoordinate2D(33.755,
 -84.39),
```

```
100 * 1609.34); // 1609.34 = meters in a mile
map.AddOverlay(circleOverlay);
```

3. Next, let's draw a polygon shape. By using a series of coordinates, you can draw any shape overlay you would like. In this simple example we are drawing the rough coordinates of the state of Wyoming on our map. Locate the comment `TODO: Step 4b – draw a polygon (Wyoming)` and uncomment the following lines:

```
var stateOverlay = MKPolygon.FromCoordinates(new CLLocationCoordinate2
D[]
    {
        new CLLocationCoordinate2D(45.00, -111.00),
        new CLLocationCoordinate2D(45, -104),
        new CLLocationCoordinate2D(41, -104),
        new CLLocationCoordinate2D(41, -111)
    });
map.AddOverlay(stateOverlay);
```

4. Finally we need to customize how these overlays are drawn. This is similar to how we customized annotations in the previous code demo by overriding `OverlayRenderer()` within our custom `MKMapViewDelegate MyMapDelegate`.Locate the comment `TODO: Step 4c – Handle how overlays are rendered` and uncomment the following lines:

```
MKCircleRenderer circleRenderer = null;
MKPolygonRenderer polyRenderer = null;
public override MKOverlayRenderer OverlayRenderer(MKMapView mapView, I
MKOverlay overlay)
{
    if (overlay is MKCircle)
    {
        if (circleRenderer == null)
        {
            circleRenderer = new MKCircleRenderer(overlay as MKCircle)
;
            circleRenderer.FillColor = UIColor.Purple;
            circleRenderer.Alpha = 0.8f;
        }
        return circleRenderer;

    } else if (overlay is MKPolygon)
    {
        if (polyRenderer == null)
        {
            polyRenderer = new MKPolygonRenderer(overlay as MKPolygon)
;
            polyRenderer.FillColor = UIColor.Green;
            polyRenderer.Alpha = 0.5f;
        }
        return polyRenderer;
    } else
    {
        Debug.WriteLine("OverlayRenderer() - Unknown overlay type!");
```

```
        return null;
    }

}
```

As you can see, we are checking to see what type of overlay it is first and then customizing how it is rendered, by specifying a different fill color and alpha transparency.

5. Run the app to see our new circle and polygon overlays.

MappingIOSLab_demo5 – Looking up addresses and nearby locations

1. Set the MappingIOSLab_demo5 project as the startup project.
2. The first thing we want to look at is how to search for an address. Locate the comment `TODO: Step 5a - search for an address` and uncomment the following method:

```
private void SearchForAddress(MKMapView mapView)
{
    var searchRequest = new MKLocalSearchRequest ();
    searchRequest.NaturalLanguageQuery = "Main Street, New York, NY";

    // perform search
    var localSearch = new MKLocalSearch (searchRequest);
    localSearch.Start (delegate (MKLocalSearchResponse response, NSErr
or error) {
        if (response != null && error == null) {
            var items = response.MapItems.ToList();
            PlotItems(mapView, items);
        } else {
            Console.WriteLine ("local search error: {0}", error);
        }
    });
}
```

3. We also want to uncomment the call to use our search method. Locate the other `TODO: Step 5a - search for an address` comment in ViewDidLoad() and uncomment the call:

```
SearchForAddress(map);
```

4. We also need to add our helper method for graphing the found places in our search. Locate the comment `TODO: Step 5b - plot found items` and uncomment the following method:

```
private void PlotItems(MKMapView mapview, List<MKMapItem> items)
{
    foreach (var item in items)
    {
        mapview.AddAnnotation(new MKPointAnnotation()
        {
```

```
            Title = item.Name,
            Coordinate = new CLLocationCoordinate2D(item.Placemark.Loc
ation.Coordinate.Latitude, item.Placemark.Location.Coordinate.Longitud
e)
        });
    }
}
```

5. Run the app and you should notice a new pinpoint right near New York, NY. If you zoom in on it closer to street level, you will notice it is actually two different pins.

6. Now lets see how we would find a bakery in a given area. Find the comment `TODO: Step 5c - Search for items of interest …` and uncomment the following method:

```
// TODO: Step 5c: Search for items of interest in the current map view
private void SearchForLocationsNearPosition(MKMapView mapView)
{
    var searchRequest = new MKLocalSearchRequest();
    searchRequest.NaturalLanguageQuery = "Bakery";
    searchRequest.Region = mapView.Region;

    var localSearch = new MKLocalSearch(searchRequest);
    localSearch.Start (delegate (MKLocalSearchResponse response, NSErr
or error)
    {
        if (response != null && error == null)
        {
            var items = response.MapItems.ToList();
            PlotItems(mapView, items);

            // zoom in on the first item we plotted
            var firstItem = items.First();
            if (firstItem != null)
            {
                MKCoordinateSpan span = new MKCoordinateSpan(0.8, 0.8)
;
                CLLocationCoordinate2D coord = new CLLocationCoordinat
e2D(
                    firstItem.Placemark.Location.Coordinate.Latitude,
                    firstItem.Placemark.Location.Coordinate.Longitude)
;
                mapView.Region = new MKCoordinateRegion(coord, span);
            }
        }
        else
        {
            Console.WriteLine ("local search error: {0}", error);
        }
    });
}
```

Notice that the query is done exactly the same, we are just specifying a region this time (whatever is currently in the `mapView`) – this is how you could search for items only within a given area (be that a physical boundary, or the current map view region).

We also added some code above that zooms the map window in on the first item that was plotted. You can see that is done above by setting the map's region. You may look into using `UIView.Animate()` when setting the region if you wanted the map to animate to the new map view location.

7. Run the app one last time toggling between using `SearchForAddress()` and `SearchForLocationsNearPosition()` and trying out different search queries.

Summary

In this lab, we showed how to use MapKit to show a map in your app and do all of the common things you might want to with a map control including graphing locations, and drawing regions. You can examine the completed version of this solution in the **MappingIOSLab_Complete** folder included in the lab resources.