**Learning Outcomes**

1. Understanding Location Services
2. Understanding Google Maps API
3. Using Maps through Intents

**Location Services**

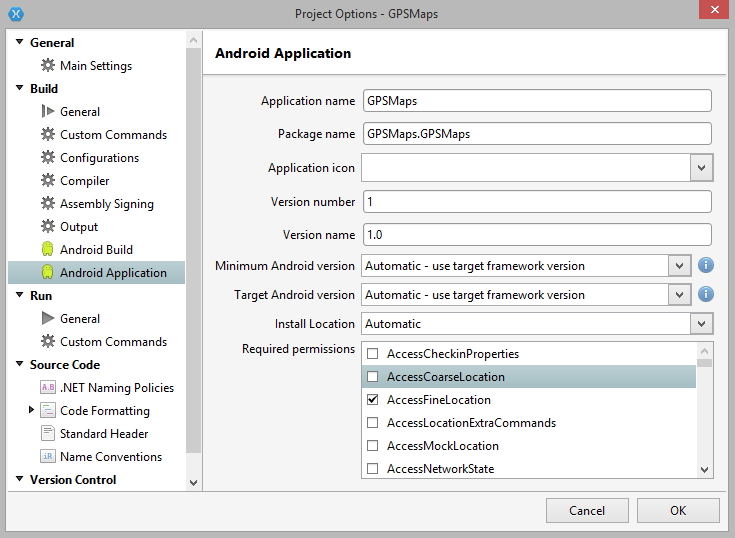
Android provides access to various location technologies such as cell tower location, Wi-Fi, and GPS. The details of each location technology are abstracted through location providers, allowing applications to obtain a location in the same way regardless of the provider used.

Location Providers

Several technologies are used internally to pinpoint the user's location. The hardware used depends on the type of location provider selected for the job of collecting data. Android uses three location providers:

1. GPS Provider – GPS gives the most accurate location, uses the most power, and works best outdoors. This provider uses a combination of GPS and assisted GPS ( [aGPS](http://en.wikipedia.org/wiki/Assisted_GPS) ), which returns GPS data collected by cellular towers.
2. Network Provider – Providers a combination of WiFi and Cellular data, including aGPS data collected by cell towers. It uses less power than the GPS Provider, but returns location data of varying accuracy.
3. Passive Provider – A "piggyback" option using providers requested by other applications or Services to generate location data in an application. This is a less reliable but power-saving option ideal for applications that don't require constant location updates to work.
4. Start a New Project -> Name it GPSMaps

Give permissions as below



Setting ACCESS\_FINE\_LOCATION implies access to both coarse and fine location data. You should never have to set both permissions, only the minimal permission your app requires to work.

The Android Location Service is the standard API for using location information on Android. Location data is collected by hardware sensors and collected by a system Service, which is accessed in the application with a LocationManager class and aILocationListener.

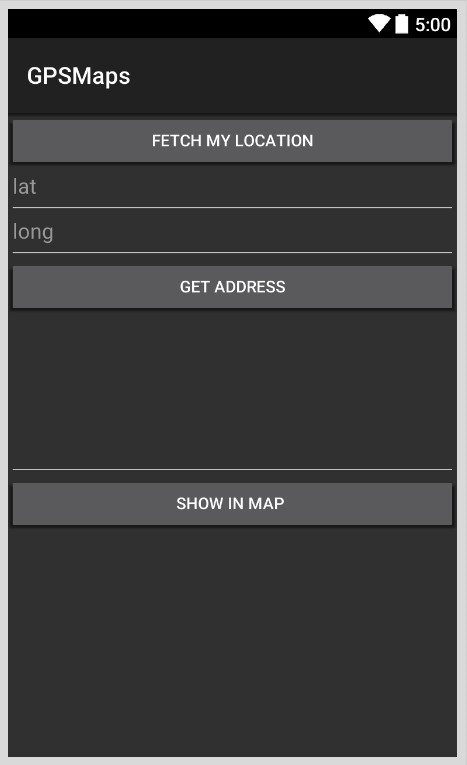
To obtain the user's location using Android Location Service, we need to:

1. Get a reference to the LocationManager class.
2. Use the LocationManager to request location updates for a specified provider.
3. Implement the ILocationListener interface and handle events when the location changes.
4. Stop location updates when application enters the background.

Location Service

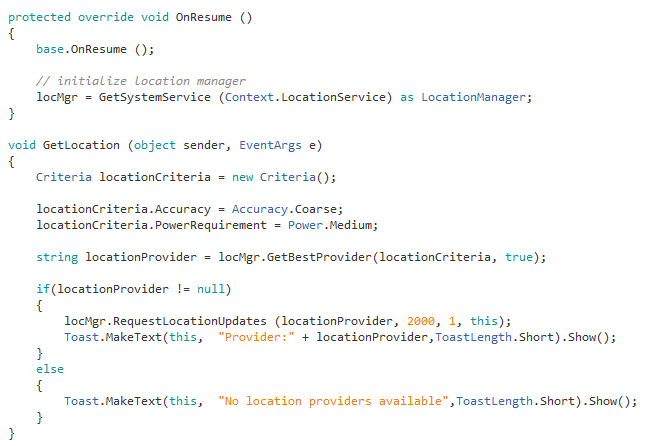
The Location Service is a special type of [Service](http://developer.android.com/guide/components/services.html) managed by the System. A System Service interacts with the device hardware, and is always running. To tap into location updates in our application, we will subscribe to location updates from the system Location Service using a LocationManager and a RequestLocationUpdates call.

**Design the app as shown below**



**MainActivity.cs**



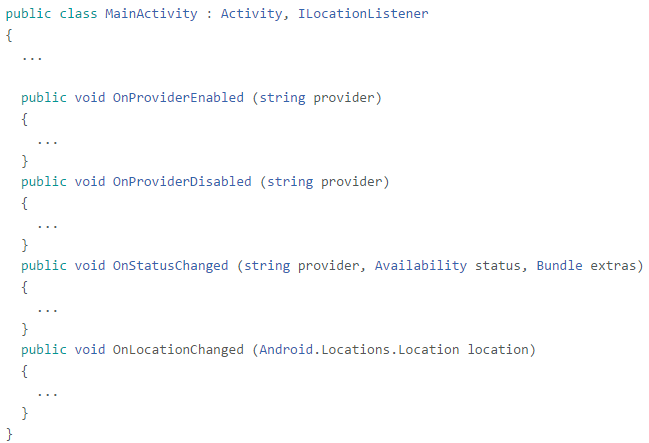




Explanation

Once an application has requested updates from the LocationManager, it can receive information from the Service by implementing the [ILocationListener](http://developer.android.com/reference/android/location/LocationListener.html" \t "_blank) interface. This interface provides methods for listening to the location Service and the location provider.

The following code sample implements ILocationListener in the *MainActivity*:



This interface lets us subscribe to four system events to check the status of the provider and get location information:

* OnProviderEnabled and OnProviderDisabled - Complementary methods that notify the application when *the user* has enabled or disabled the provider (for example, a user may disable GPS to conserve battery).
* OnStatusChanged - Notifies the application when the provider's availability changes, and provides the accompanying status (for example, GPS availability may change when a user walks indoors).
* OnLocationChanged - The System will call OnLocationChanged when the user's location changes enough to qualify as a location change according to the Criteria we set when requesting location updates.

## Stop Location Updates

The RemoveUpdates method tells the system location Service to stop sending updates to our application. By calling this in OnPause, we are able to conserve power if an application doesn’t need location updates while its Activity is not on the screen:

protected override void OnPause ()

{

base.OnPause ();

locMgr.RemoveUpdates (this);

}

If your application needs to get location updates while in the background, you'll want to create a custom Service that subscribes to the system Location Service. Refer to the [Backgrounding](http://docs.xamarin.com/guides/cross-platform/application_fundamentals/backgrounding/part_4_ios_backgrounding_walkthroughs/location_walkthrough/) guide and sample application for more information.

## GetBestProvider

The application above sets GPS as the location provider. However, GPS may not be available in all cases, such as if the device is indoors or does not have a GPS receiver. If this is the case, we will get a null return for the Provider. If we want our app to work when GPS is not available, we can use the GetBestProvider method to ask for the best available (device-supported and user-enabled) location provider at application launch. Instead of passing in a specific provider, we can tell GetBestProvider our requirements for the provider - such as accuracy and power - with a [Criteria object](http://developer.android.com/reference/android/location/Criteria.html).GetBestProvider returns the best provider for the given Criteria.

**Reverse Geo-Coding**

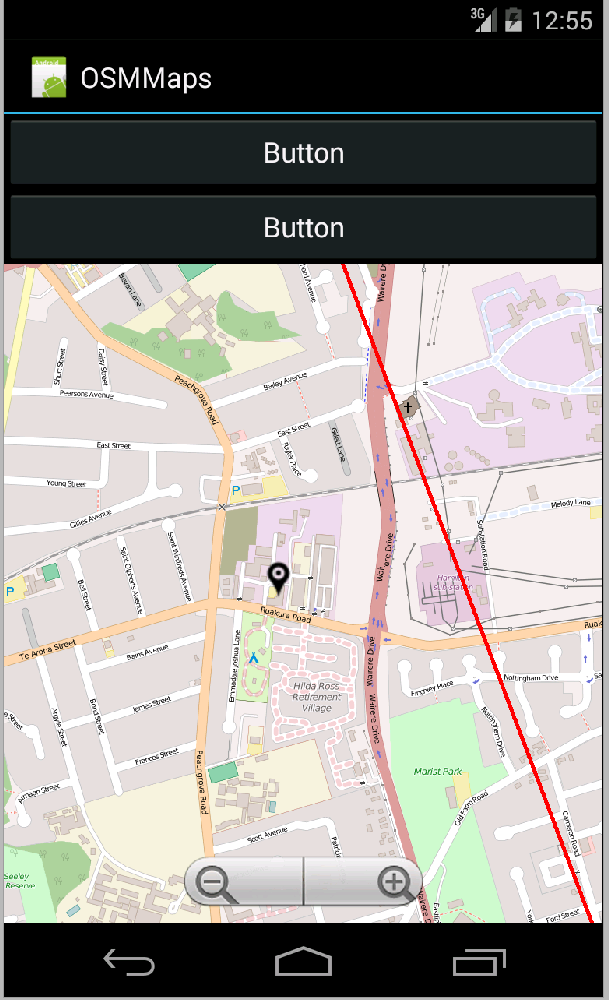
This recipe creates a Geocoder instance, which is in the Android.Locations namespace. The Geocoder callsGetFromLocationAsync with the latitude and longitude to reverse geocode. This will perform the network call asynchronously so that main UI thread is not blocked. When the result is returned, the method will return a list of addresses and continue on the UI thread.

**Open Street Maps**

**Learning Outcomes**

1. Displaying maps using Open Street Maps

**Screenshot**



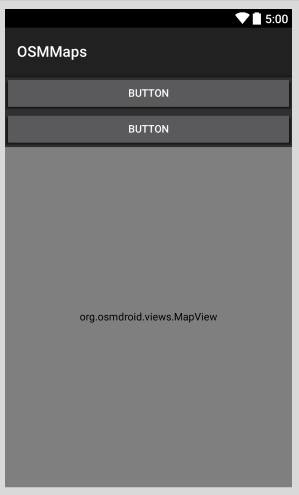
Open Street maps is another component that can be used to display maps and markers. It is a good alternative to google maps as it does not require an api key and its free to use.

Create a new project call it OSMMaps.

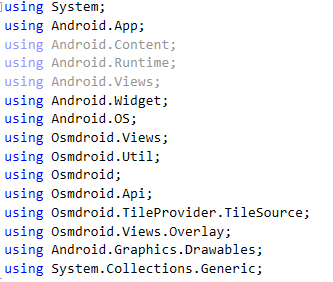
Add the component



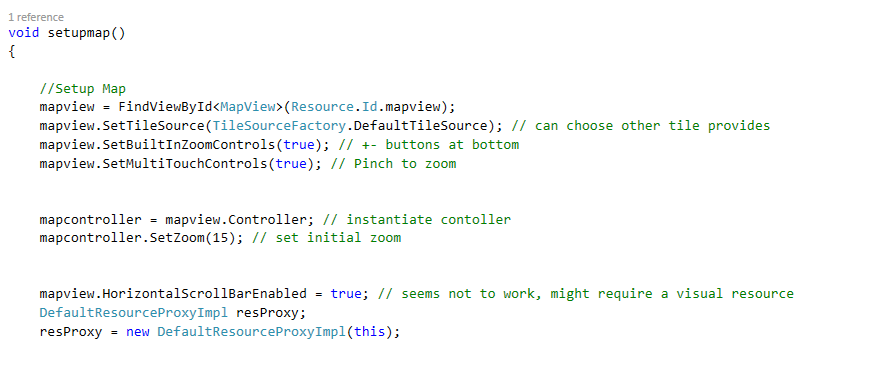
Create the following layout

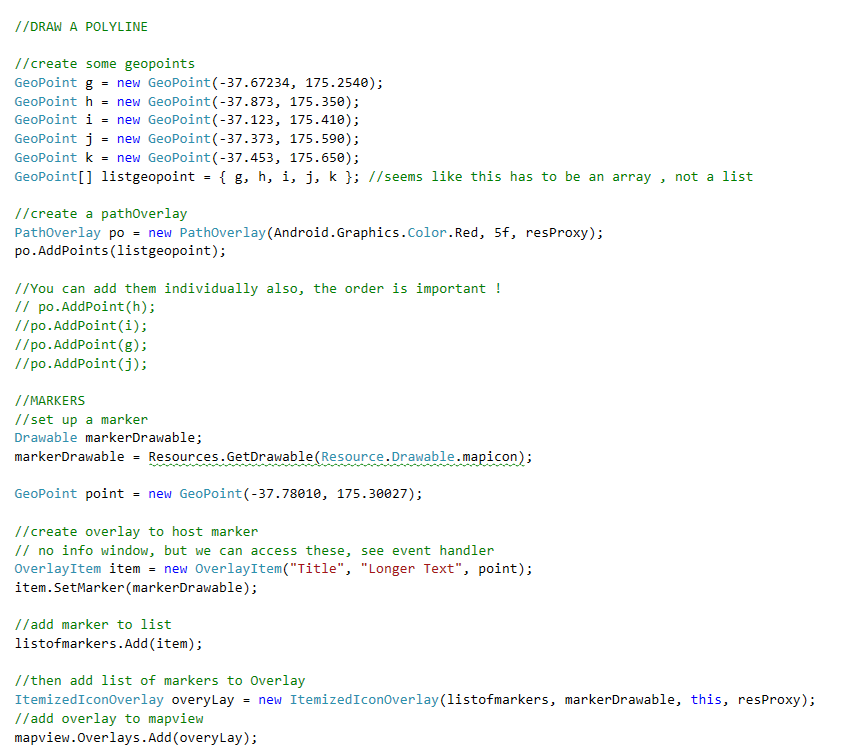


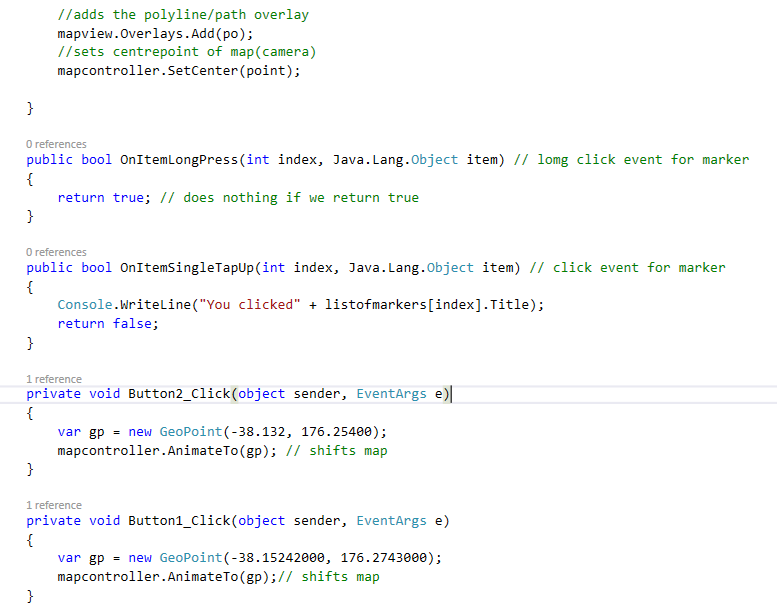
MainActivity.cs



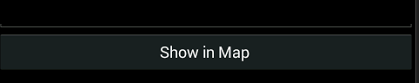








**Google Maps API**



On click of the Show in Map button we want to open up another activity and a show a map and add a marker on the map.

For this we would be using the Google Maps API

Follow the steps at the links shown below

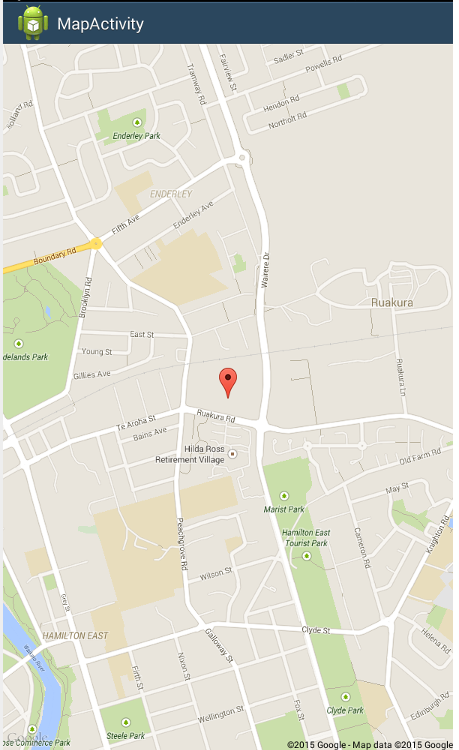
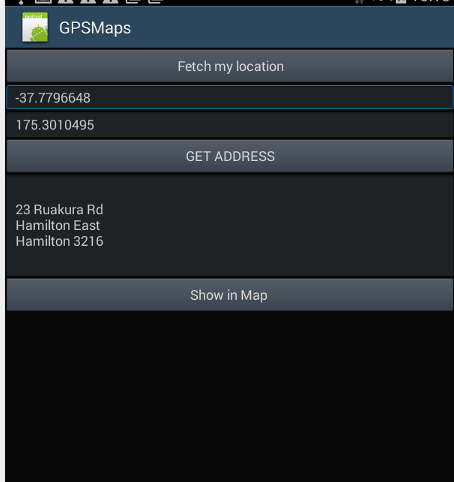
**Google maps API**

<https://developer.xamarin.com/guides/android/platform_features/maps_and_location/maps/part_2_-_maps_api/>

**Obtaining the Google Api key**

<https://developer.xamarin.com/guides/android/platform_features/maps_and_location/maps/obtaining_a_google_maps_api_key/>

Final Output



There are a series of steps to be performed and some of them are very important

Add the google play services component from the component store



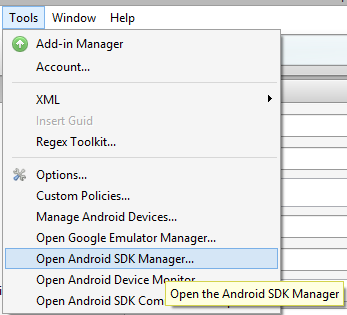
1. Add a new layout call it Map.axml

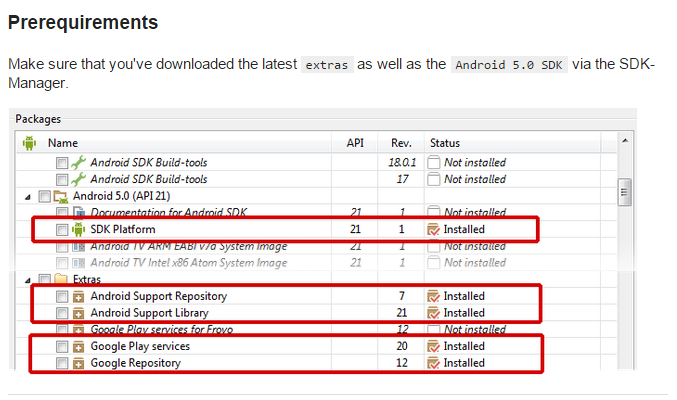


Map.axml

<?xml version="1.0" encoding="utf-8"?>  
<fragment xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/map"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    class="com.google.android.gms.maps.MapFragment" />

1. Adding Google Play services from the SDK Manager





1. Edit your AndroidManifest.xml



\*\*\*\*\*\*\*\*\* Note : **Your Package name should be in lowercase** (very important)



Set your Minimum Android version to the phone or Emulator your using, set the Target to the latest Android version (in this case 5.0)



Go to the source section of your android manifest



Copy down the further permissions required in the android manifest file.

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" package="gpsmaps.gpsmaps">  
    <uses-sdk android:minSdkVersion="19" android:targetSdkVersion="21" />  
    <application android:label="GPSMaps">  
        <meta-data android:name="com.google.android.maps.v2.API\_KEY" android:value="YOUR\_KEY" />  
        <meta-data android:name="com.google.android.gms.version" android:value="@integer/google\_play\_services\_version" />  
    </application>  
    <permission android:name="gpsmaps.permission.MAPS\_RECEIVE" android:protectionLevel="signature" />  
    <uses-permission android:name="gpsmaps.permission.MAPS\_RECEIVE" />  
    <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />  
    <uses-feature android:glEsVersion="0x00020000" android:required="true" />  
    <uses-permission android:name="android.permission.INTERNET" />  
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ\_GSERVICES" />  
    <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />  
    <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />  
    <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />  
    <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />  
</manifest>

API key from google developers, see below steps to get a key

Change it to your package name

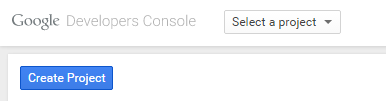
**Obtaining a Google API Key**

Login to your Google account

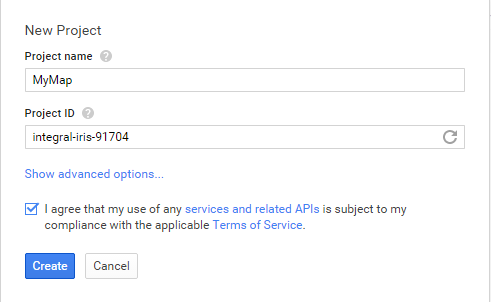
Go to the google developers console at

<https://console.developers.google.com/>

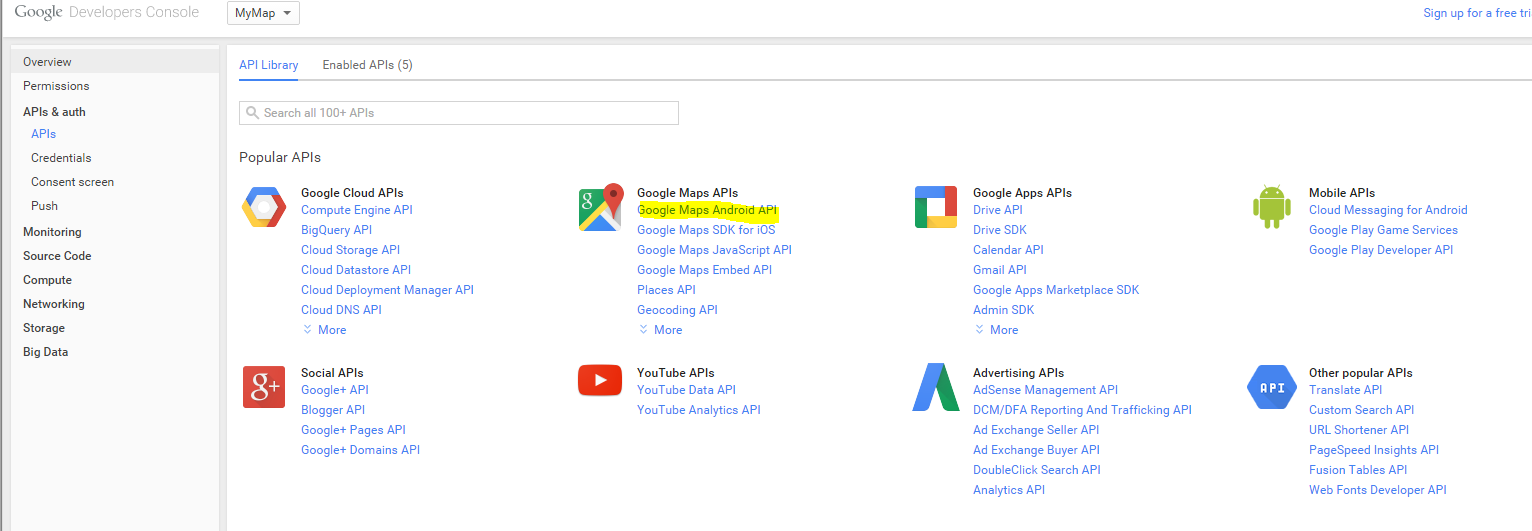
Click on Create Project



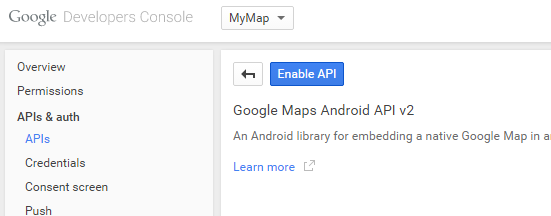
Create a new Project



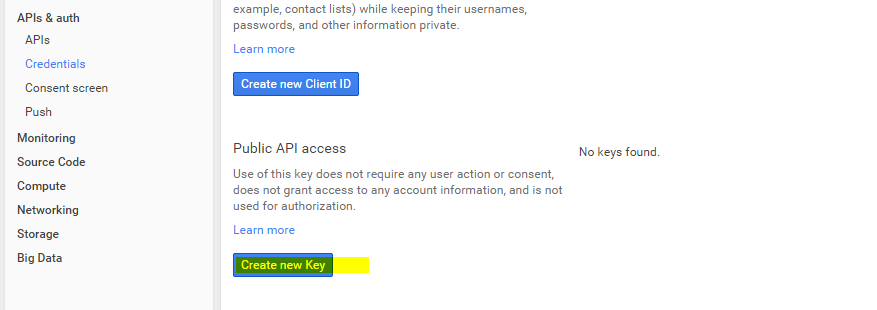
Select your new project and in the google developer console search for Google Maps Android API.



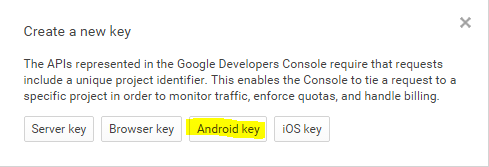
Enable the API

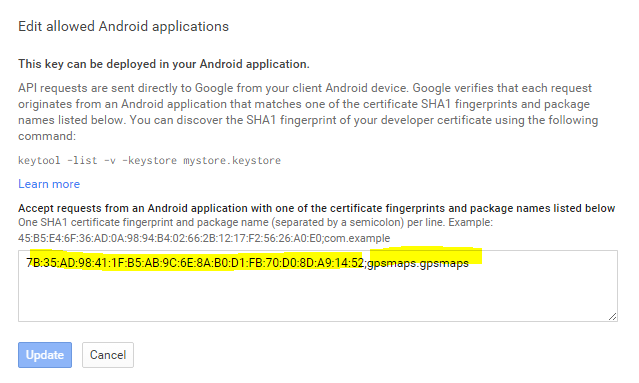


Click on Credentials and Create New Key



Create an Android Key

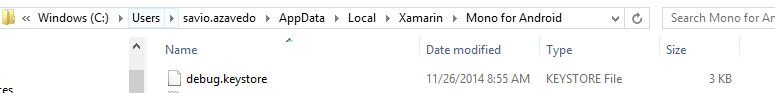




In the screen above we need to enter the SHA1 fingerprint from debug keystore file used by our app (or signed key if uploading to the app store) and the package name (Should be in lowercase important!)

To obtain the debug key follow the steps below.

By default Xamarin debug keystore file resides in



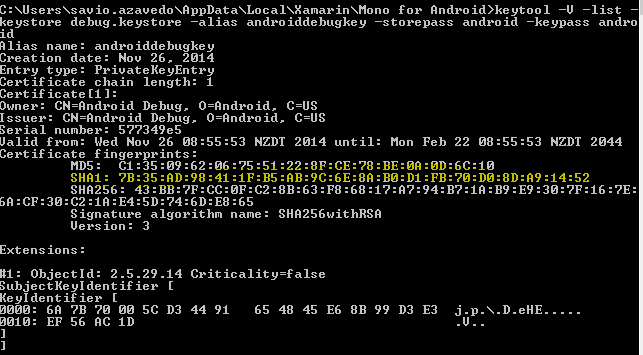
Open Command Prompt

Run -> CMD

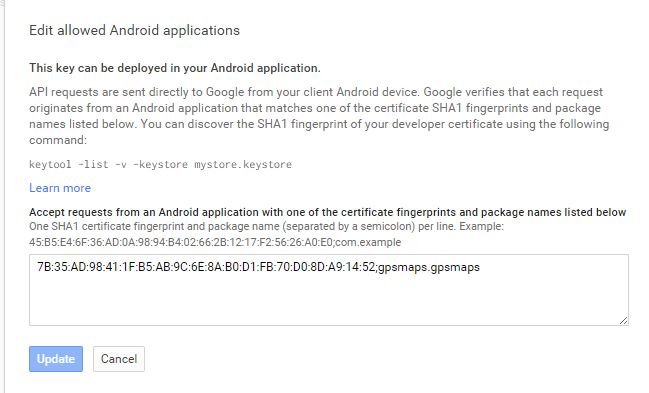
Go to

cd C:\Program Files\Java\jre7\bin

keytool -V -list -keystore "C:\Users\<Your\_Username>\AppData\Local\Xamarin\Mono for Android\debug.keystore" -alias androiddebugkey -storepass android -keypass android

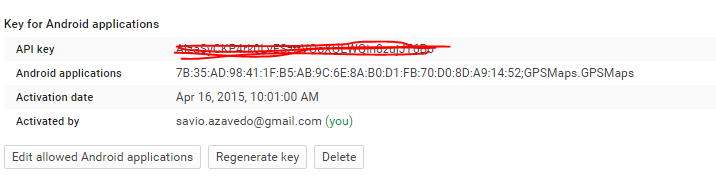


You should see the SHA1 Fingerprint, copy it and paste it



And click Save (or Update)

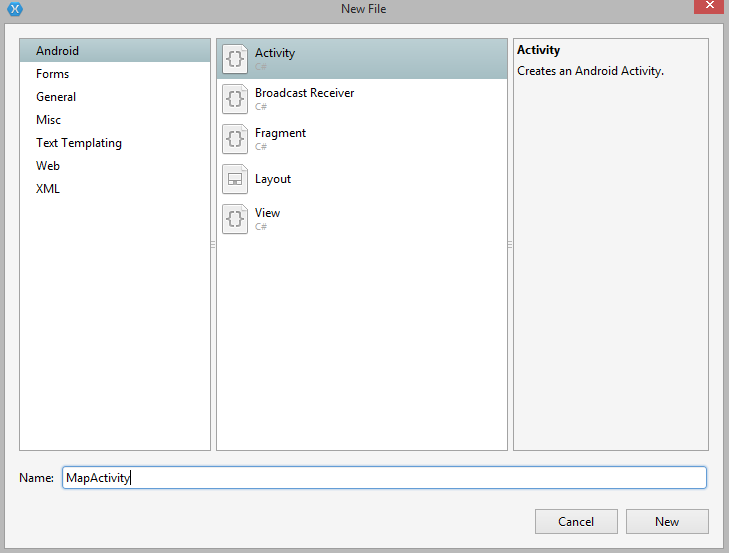
Finally you get the key for Android Applications.

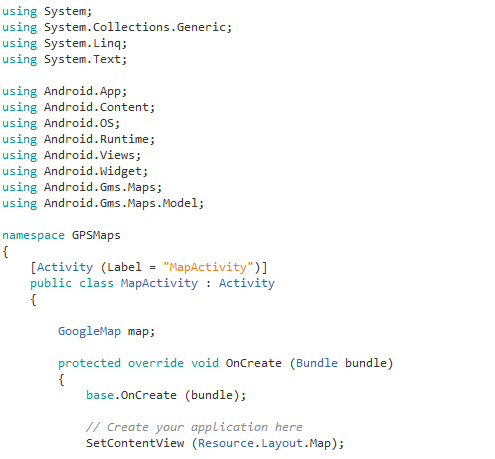


Copy the API key and paste it in the Manifest.xml



Create a new Activity call it MapActivity.cs, call the SetContentView to display the Map layout.



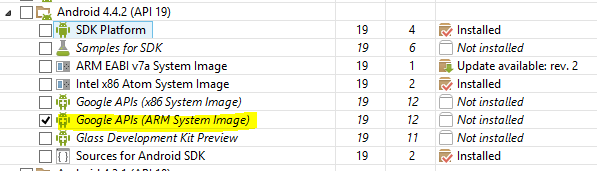


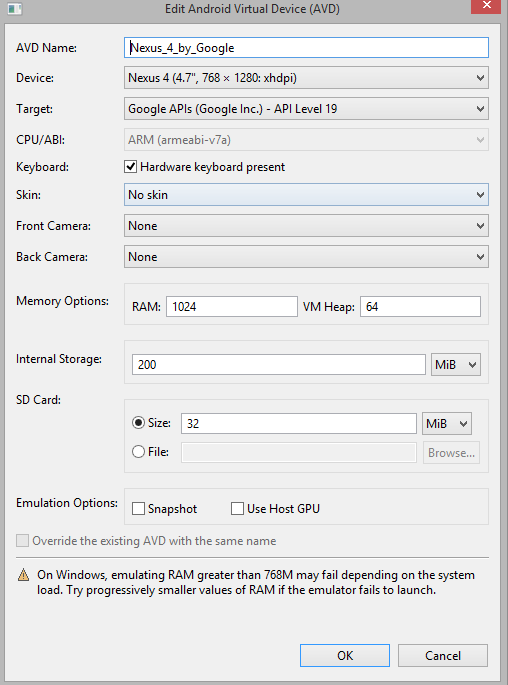
Build the project and hopefully it should not give you any errors. !!!!

Running the app

* 1. On the device - It should work fine
  2. On the Emulator - you need to create a new emulator which supports Google API’s (steps below)

In the SDK manager for the Android 4.4 version download the Google API’s (ARM system Image) and create the Virtual Device as shown.



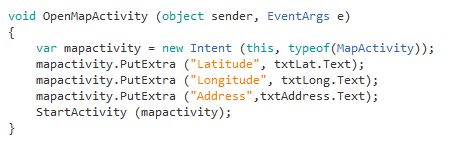


Run the app and check the result.

Call startActivity(typeof(MapActivity)) on the click of OpenMapActivity button.

**Transferring the co-ordinates and showing a marker on the Map**

MainActivity.cs



MapActivity.cs

