# Making Your App Location-Aware

One of the unique features of mobile applications is location awareness. Mobile users take their devices with them everywhere, and adding location awareness to your app offers users a more contextual experience.

The location APIs available in Android API and Google Play services that facilitate adding location awareness to your app with automated location tracking, geofencing, and activity recognition.

The [Google Play services location APIs](http://developer.android.com/reference/com/google/android/gms/location/package-summary.html) are preferred over the Android framework location APIs ([android.location](http://developer.android.com/reference/android/location/package-summary.html)) as a way of adding location awareness to your app. If you are currently using the Android framework location APIs, you are strongly encouraged to switch to the Google Play services location APIs as soon as possible.

**ANDROID LOCATION API**

Android.jar (framework) has Location API in the form of Android.location package .

Location API contains some predefined classes/interfaces and their respective methods and member that provide facilities to determine the current geo position of device.

NOTE:- Most Android devices have sensor which allow to determine the current geo location.

* This can be done either via GPS (Global positioning System) or via cell tower n/w or via wi-fi Networks.
* Android.jar (framework) has Location API in the form of Android.location package .

Example- Few classes can be …

|  |  |
| --- | --- |
| [Location](http://developer.android.com/reference/android/location/Location.html) | A data class representing a geographic location. |
| [LocationManager](http://developer.android.com/reference/android/location/LocationManager.html) | This class provides access to the system location services. |
| [LocationProvider](http://developer.android.com/reference/android/location/LocationProvider.html) | An abstract superclass for different location providers. |
| Criteria | A class indicating the application criteria for selecting a location provider. |

**Interface can be ….**

|  |  |
| --- | --- |
| [LocationListener](http://developer.android.com/reference/android/location/LocationListener.html) | Used for receiving notifications from the LocationManager when the  location has changed. |

**LocationManager Class:**-

An object of this class provide access to the Android location service and This services allows to access location providers, to register location update listeners and proximity alerts and more.

**Location** Class:- An Object of this class stores Geographical co-ordinates (Latitude and Longitude).

**Location Provider class**:

Location Provider Class is the super class of different Location Provider(Classes) which deliver the information about the current location.

This information stores in the object of “**Location**” Class.

The Android device might have one or several location provider available and we can select accordingly to our need.

* Most devices are having following location provider available-

1. Network: It uses the mobile network or wifi to determine the best location.
2. GPS: uses the GPS receiver in the android device to determine the best location with the help of satellite.

**How to select best LocationProvider ??**

For selection of best Location Provider we have to use an object of **Criteria class.**

Note:- if we want to access GPS censor ( Location Service) , we need to take following permission in Manifest file.

*<use permission android: name=”android.permission.ACCESS\_FINE\_LOCATION”/>*

Note:-Apart from this we have to take internet permission as well.

*<use –permission android: name =”android.permission.INTERNET”/>*

**Prompt the user to Enabled GPS**

The user can decide if the GPS is enabled or not.

You can find out, if a LocationManager is enabled via the isProviderEnabled() method. If its not enabled you can send the user to the settings via an Intent with the Settings.ACTION\_LOCATION\_SOURCE\_SETTINGS action for the android.provider.Settings class.

LocationManager service = (LocationManager) getSystemService(LOCATION\_SERVICE);

**boolean** enabled = service

.isProviderEnabled(LocationManager.GPS\_PROVIDER);

*// check if enabled and if not send user to the GSP settings*

*// Better solution would be to display a dialog and suggesting to*

*// go to the settings*

**if** (!enabled) {

Intent intent = **new** Intent(Settings.ACTION\_LOCATION\_SOURCE\_SETTINGS);

startActivity(intent);

}

Note:- We can register **LocationListerner** object with **LocationManager** object to receive periodic update about geo position.

**LocationListener** :- It is an interface and implementation of this interface is used for receiving notifications from the LocationManager when the location has changed.

**Event Handler methods:-**

1-**void onLocationChanged(**Location location**)**- called by OS when location changes.

2- **void onProviderDisabled(**String providerName**)**- called by OS when provider has been disabled by the user.

3- **void onProviderEnabled(**String providerName**)**- called by OS when provider has been Enabled by the user..

4- **void onStatusChanged(**Stringprovider,int status,Bundle extra**)**- called by OS when provider status changes.

Where -

|  |  |
| --- | --- |
| *Provider* | the name of the location provider associated with this update. |
| *Status* | [OUT\_OF\_SERVICE](http://developer.android.com/reference/android/location/LocationProvider.html#OUT_OF_SERVICE) if the provider is out of service, and this is not expected to change in the near future; [TEMPORARILY\_UNAVAILABLE](http://developer.android.com/reference/android/location/LocationProvider.html#TEMPORARILY_UNAVAILABLE) if the provider is temporarily unavailable but is expected to be available shortly; and [AVAILABLE](http://developer.android.com/reference/android/location/LocationProvider.html#AVAILABLE) if the provider is currently available. |
| *Extras* | An optional Bundle which will contain provider specific status variables. |

These methods are called if the LocationListener has been registered with the location manager service using the following method.

public void [requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)](http://developer.android.com/reference/android/location/LocationManager.html#requestLocationUpdates(java.lang.String, long, float, android.location.LocationListener));

Where –

provider the name of the provider with which to register

minTime minimum time interval between location updates, in milliseconds

minDistance minimum distance between location updates, in meters

[LocationListener](http://developer.android.com/reference/android/location/LocationListener.html)  Listener Object

**NOTE:- This API is not the recommended method for accessing Android location.**  
The [Google Location Services API](http://developer.android.com/reference/com/google/android/gms/location/package-summary.html) is a part of **Google Play services**, is the preferred way to add location-awareness to your app. It offers a simpler API, higher accuracy, low-power geofencing, and more. If you are currently using the android.location API, you are strongly encouraged to switch to the Google Location Services API as soon as possible.

**Android Fused Location Provider API**

This Android tutorial is to explain what a fused location provider and how to use it to get the location using a sample Android application. We need not explicitly choose either GPS or Network location Provider, as the “Fused Location Provider” automatically chooses the underlying technology and gives the best location as per the need.

In this tutorial we will be using the FusedLocationProviderApi which is **the latest API and the best among the available possibilities to get location in Android**.

This Android tutorial shows you how to use the Google Play services location APIs in your app to get the current location, get periodic location updates, and look up addresses.

## Fused Location Provider

* It provides simple and easy to use APIs.
* Provides high accuracy over other options.
* Utilizes low power by choosing the most efficient way to access the location.

## Example Android Application to Get Location

Following is an example Android application that shows how to get the current user location. This also continuously updates the location on the move.

### Prerequisite

Google Play Services is required. If you are using Android Studio and Gradle, you should have the following dependencies added in build.gradle file

dependencies {

compile fileTree(dir: 'libs', include: ['\*.jar'])

compile 'com.android.support:appcompat-v7:21.0.3'

compile 'com.google.android.gms:play-services:8.4.0'

}

***Note:****As of now(14 March 2015), Fused location provider API is not working in all the Android Nexus virtual device emulators. Its a widely reported defect. You can use any other AVD to test in the emulator like WVGA AVDs. I used “4 WVGA API 21” virtual device in emulator to test it. It is also working on a real device and I tested it in Panasonic P81.*

### AndroidManifest.xml

Remember to give access permissions as shown below. You need not give permission for GPS, Network providers. I have seen Android location access tutorial examples giving all the available permission in the world. Just on android.permission.ACCESS\_FINE\_LOCATION is enough.

ACCESS\_COARSE\_LOCATION permission is for approximate location access using cell towers and Wi-Fi. ACCESS\_FINE\_LOCATION is for precise location access using GPS or cell towers and Wi-Fi. Choose the one that is appropriate for you and do not give both. Battery power is used accordingly.

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.javapapers.android.androidlocation" >

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme" >

<activity

android:name=".LocationActivity"

android:label="@string/app\_name" >

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<meta-data

android:name="com.google.android.gms.version"

android:value="@integer/google\_play\_services\_version" />

</application>

</manifest>

## FusedLocationProviderApi Usage

### FusedLocationProviderApi

[FusedLocationProviderApi](https://developer.android.com/reference/com/google/android/gms/location/FusedLocationProviderApi.html" \o "FusedLocationProviderApi API" \t "_blank) can be accessed via LocationServices as below,

private FusedLocationProviderApi fusedLocationProviderApi = LocationServices.FusedLocationApi;

## Three Use Cases for location access

With respect to fused location provider (Object), we can broadly classify the API usage in three use cases.

1. **getLastLocation(GoogleApiClient)** this API should be used when there is **no need for continuous access** to location from an application. Like one shot access or get user location based on some action. This is the simplified way to get the device location and also may not provide high accuracy.
2. **requestLocationUpdates(GoogleApiClient,LocationRequest, LocationListener)** this API should be used when there a **need for continuous location updates** and the location is accessed when the application is active in foreground.

### requestLocationUpdates (GoogleApiClient, LocationRequest, PendingIntent) this API is used to **receive location updates in the background** even when the application is not active. So the difference is PendingIntent.

### Then we will invoke any of above method on fusedLocationProviderApi to register Location listener and get PendingResult<R extends Result>.

### Eg.

### PendingResult<Status> pendingResult = LocationServices.FusedLocationApi.requestLocationUpdates(

### mGoogleApiClient, mLocationRequest, this);

### where-

### Parameters are-

### GoogleApiClient

FusedLocationProviderApi requires the GoogleApiClient instance to get the Location and it can be obtained as below.

googleApiClient = new GoogleApiClient.Builder(locationActivity)

.addApi(LocationServices.API)

.addConnectionCallbacks(this)

.addOnConnectionFailedListener(this)

.build();

* Instantiating the GoogleApiClient should be done in onCreate of the application Activity.
* googleApiClient.connect() should be done in onStart and disconnect() in onStop.
* When the GoogleApiClient is connected successfully,onConnected(Bundle) callback will be called. There we should register with LocationListener for location updates as LocationServices.FusedLocationApi.requestLocationUpdates(  
  mGoogleApiClient, mLocationRequest, this);

**LocationRequest –**

private static final long INTERVAL = 1000 \* 10;

private static final long FASTEST\_INTERVAL = 1000 \* 5;

LocationRequest mLocationRequest = new LocationRequest();

mLocationRequest.setInterval(INTERVAL);

mLocationRequest.setFastestInterval(FASTEST\_INTERVAL);

mLocationRequest.setPriority(LocationRequest.PRIORITY\_HIGH\_ACCURACY);

**LocationListener -** Used for receiving notifications from the [FusedLocationProviderApi](https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderApi.html) when the location has changed.

### Returns –

**PendingResult<R> -**

* a PendingResult for the call, check [isSuccess()](https://developers.google.com/android/reference/com/google/android/gms/common/api/Status.html" \l "isSuccess()) to determine if it was successful.

### Listener Interfaces Implementations for Callbacks

The following interfaces should be implemented to get the location update.

* LocationListener provides call back for location change through onLocationChanged.
* GoogleApiClient.ConnectionCallbacks provides call back for GoogleApiClient onConnected.
* GoogleApiClient.OnConnectionFailedListener provides call back for GoogleApiClient onConnectionFailed.

## Test Run the Location Application via Android Emulator

In Eclipse, Open DDMS perspective (Window -> Open Perspective)

* Select your emulator device
* Select the tab named emulator control
* In ‘Location Controls’ panel, ‘Manual’ tab, give the Longitude and Latitude as input and ‘Send’.

# Adding Maps

Android allows us to integrate Google maps in our application. You can show any location on the map, or can show different routes on the map etc. You can also customize the map according to your choices.



The [Google Maps Android API](https://developers.google.com/maps/documentation/android/) allows you to include maps and customized mapping information in your app.

## Integrating Google Maps -

Integrating Google maps in your application basically consists of these 4 steps.

* Download and configure. Google Play Services SDK
* Create a Google Maps project
* Obtain API key from Google console
* Specify Android Manifest settings

## Create a Google Maps project

Follow these steps to create a new app project including a map activity:

1. Start Android Studio.
2. Create a new project as follows:
   * If you see the **Welcome to Android Studio** dialog, choose **Start a new Android Studio project**, available under 'Quick Start' on the right of the dialog.
3. Enter your app name, company domain, and project location, as prompted. Then click **Next**.
4. Select the form factors you need for your app. If you're not sure what you need, just select **Phone and Tablet**. Then click **Next**.
5. Select **Google Maps Activity** in the 'Add an activity to Mobile' dialog. Then click **Next**.
6. Enter the activity name, layout name and title as prompted. The default values are fine. Then click **Finish**.

Note- Android Studio starts Gradle and builds your project. This may take a few seconds.

Note- When the build is finished, Android Studio opens the google\_maps\_api.xml and the MapsActivity.java files in the editor. (Note that your activity may have a different name, but it will be the one you configured during setup.) Notice that the google\_maps\_api.xml file contains instructions on getting a Google Maps API key before you try to run the application. The next section describes getting the API key in more detail.

## Get a Google Maps API key

Your application needs an API key to access the Google Maps servers. The type of key you need is a **Key for Android applications**. The key is free. You can use it with any of your applications (if is having same packageName)that call the Google Maps Android API, and it supports an unlimited number of users.

Choose **one of the following ways** to get your API key:

* **The fast, easy way:** Use the link provided in the google\_maps\_api.xml file that Android Studio created for you:
  1. Copy the link provided in the google\_maps\_api.xml file and paste it into your browser. The link takes you to the Google Developers Console and supplies information via URL parameters, thus reducing the manual input required from you.
  2. Follow the instructions to create a new project on the console or select an existing project.
  3. Create an Android API key for your console project.
  4. Copy the resulting API key, go back to Android Studio, and paste the API key into the <string> element in the google\_maps\_api.xml file.
* **A slightly less fast way:** Use the credentials provided in the google\_maps\_api.xml file that Android Studio created for you:
  1. Copy the credentials provided in the google\_maps\_api.xml file.
  2. Go to the [Google Developers Console](https://console.developers.google.com/) in your browser.
  3. Use the copied credentials to add your app to an existing API key or to create a new API key.
* **The full process for getting an API key:** If neither of the above options works for your situation, follow the [complete process](https://developers.google.com/maps/documentation/android-api/signup).

## C:\Users\Mokib Khan\Desktop\1.png

## C:\Users\Mokib Khan\Desktop\2.png

## C:\Users\Mokib Khan\Desktop\3.png

## C:\Users\Mokib Khan\Desktop\4.png

## C:\Users\Mokib Khan\Desktop\6.png

## Hello Map! Take a look at the code

Examine the code supplied by the template. In particular, look at the following files in your Android Studio project.

### The XML layout file

By default, the XML file that defines the app's layout is at res/layout/activity\_maps.xml. It contains the following code:

<fragment xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"  
    android:id="@+id/map"  
    tools:context=".MapsActivity"  
    android:name="com.google.android.gms.maps.SupportMapFragment" />

### The maps activity Java file

By default, the Java file that defines the maps activity is named MapsActivity.java. It should contain the following code after your package name:

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {  
  
    private GoogleMap mMap;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity\_maps);  
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()  
                .findFragmentById(R.id.map);  
        mapFragment.getMapAsync(this);  
    }  
  
    @Override  
    public void onMapReady(GoogleMap googleMap) {  
        mMap = googleMap;  
  
        // Add a marker in Sydney, Australia, and move the camera.  
        LatLng sydney = new LatLng(-34, 151);  
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));  
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));  
    }  
}

If your maps activity file doesn't contain the above code, replace the contents of the file with the above code, after your package name.