Integrating GPS in mobile application

Android location APIs make it easy for you to build location-aware applications, without needing to focus on the details of the underlying location technology.

This becomes possible with the help of **Google Play services**, which facilitates adding location awareness to your app with automated location tracking, geofencing, and activity recognition.

This tutorial shows you how to use Location Services in your APP to get the current location, get periodic location updates, look up addresses etc.

The Location Object

The **Location** object represents a geographic location which can consist of a latitude, longitude, time stamp, and other information such as bearing, altitude and velocity. There are following important methods which you can use with Location object to get location specific information –

Sr.No.	Method & Description
1	**float distanceTo(Location dest)** Returns the approximate distance in meters between this location and the given location.
2	**float getAccuracy()** Get the estimated accuracy of this location, in meters.
3	**double getAltitude()** Get the altitude if available, in meters above sea level.
4	**float getBearing()**

	Get the bearing, in degrees.
5	**double getLatitude()** Get the latitude, in degrees.
6	**double getLongitude()** Get the longitude, in degrees.
7	**float getSpeed()** Get the speed if it is available, in meters/second over ground.
8	**boolean hasAccuracy()** True if this location has an accuracy.
9	**boolean hasAltitude()** True if this location has an altitude.
10	**boolean hasBearing()** True if this location has a bearing.
11	**boolean hasSpeed()** True if this location has a speed.
12	**void reset()** Clears the contents of the location.
13	**void setAccuracy(float accuracy)** Set the estimated accuracy of this location, meters.

14	**void setAltitude(double altitude)** Set the altitude, in meters above sea level.
15	**void setBearing(float bearing)** Set the bearing, in degrees.
16	**void setLatitude(double latitude)** Set the latitude, in degrees.
17	**void setLongitude(double longitude)** Set the longitude, in degrees.
18	**void setSpeed(float speed)** Set the speed, in meters/second over ground.
19	**String toString()** Returns a string containing a concise, human-readable description of this object.

Get the Current Location

To get the current location, create a location client which is **LocationClient** object, connect it to Location Services using **connect()** method, and then call its **getLastLocation()** method. This method returns the most recent location in the form of **Location** object that contains latitude and longitude coordinates and other information as explained above. To have location based functionality in your activity, you will have to implement two interfaces –

- GooglePlayServicesClient.ConnectionCallbacks
- GooglePlayServicesClient.OnConnectionFailedListener

These interfaces provide following important callback methods, which you need to implement in your activity class −

Sr.No.	Callback Methods & Description
1	**abstract void onConnected(Bundle connectionHint)** This callback method is called when location service is connected to the location client successfully. You will use **connect()** method to connect to the location client.
2	**abstract void onDisconnected()** This callback method is called when the client is disconnected. You will use **disconnect()** method to disconnect from the location client.
3	**abstract void onConnectionFailed(ConnectionResult result)** This callback method is called when there was an error connecting the client to the service.

> You should create the location client in **onCreate()** method of your activity class, then connect it in **onStart()** , so that Location Services maintains the current location while your activity is fully visible. You should disconnect the client in **onStop()** method, so that when your app is not visible, Location Services is not maintaining the current location. This helps in saving battery power up-to a large extent.

Get the Updated Location

If you are willing to have location updates, then apart from above mentioned interfaces, you will need to implement **LocationListener** interface as well. This interface provide following callback method, which you need to implement in your activity class −

Sr.No.	Callback Method & Description
1	**abstract void onLocationChanged(Location location)** This callback method is used for receiving notifications from the LocationClient when the location has changed.

Location Quality of Service

The **LocationRequest** object is used to request a quality of service (QoS) for location updates from the **LocationClient**. There are following useful setter methods which you can use to handle QoS. There are equivalent getter methods available which you can check in Android official documentation.

Sr.No.	Method & Description
1	**setExpirationDuration(long millis)** Set the duration of this request, in milliseconds.
2	**setExpirationTime(long millis)** Set the request expiration time, in millisecond since boot.
3	**setFastestInterval(long millis)** Explicitly set the fastest interval for location updates, in milliseconds.
4	**setInterval(long millis)** Set the desired interval for active location updates, in milliseconds.
5	**setNumUpdates(int numUpdates)**

	Set the number of location updates.
6	**setPriority(int priority)** Set the priority of the request.

Now for example, if your application wants high accuracy location it should create a location request with **setPriority(int)** set to PRIORITY_HIGH_ACCURACY and **setInterval(long)** to 5 seconds. You can also use bigger interval and/or other priorities like PRIORITY_LOW_POWER for to request "city" level accuracy or PRIORITY_BALANCED_POWER_ACCURACY for "block" level accuracy.

> Activities should strongly consider removing all location request when entering the background (for example at onPause()), or at least swap the request to a larger interval and lower quality to save power consumption.

Displaying a Location Address

Once you have **Location** object, you can use **Geocoder.getFromLocation()** method to get an address for a given latitude and longitude. This method is synchronous, and may take a long time to do its work, so you should call the method from the **doInBackground()** method of an **AsyncTask** class.

The **AsyncTask** must be subclassed to be used and the subclass will override **doInBackground(Params...)** method to perform a task in the background and **onPostExecute(Result)** method is invoked on the UI thread after the background computation finishes and at the time to display the result. There is one more important method available in AyncTask which is **execute(Params... params)**, this method executes the task with the specified parameters.

Example

Following example shows you in practical how to to use Location Services in your app to get the current location and its equivalent addresses etc.

> To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Create Android Application

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *Tutorialspoint* under a package *com.example.tutorialspoint7.myapplication*.
2	add *src/GPSTracker.java* file and add required code.
3	Modify *src/MainActivity.java* file and add required code as shown below to take care of getting current location and its equivalent address.
4	Modify layout XML file *res/layout/activity_main.xml* to add all GUI components which include three buttons and two text views to show location/address.
5	Modify *res/values/strings.xml* to define required constant values
6	Modify *AndroidManifest.xml* as shown below
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **MainActivity.java**.

```
package com.example.tutorialspoint7.myapplication;
import            android.Manifest;import            android.app.Activity;import
android.os.Bundle;import          android.support.v4.app.ActivityCompat;import
android.test.mock.MockPackageManager;import          android.view.View;import
android.widget.Button;import android.widget.Toast;
public class MainActivity extends Activity {

  Button btnShowLocation;
  private static final int REQUEST_CODE_PERMISSION = 2;
  String mPermission = Manifest.permission.ACCESS_FINE_LOCATION;

  // GPSTracker class
  GPSTracker gps;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    try {
      if (ActivityCompat.checkSelfPermission(this, mPermission)
        != MockPackageManager.PERMISSION_GRANTED) {

        ActivityCompat.requestPermissions(this, new String[]{mPermission},
          REQUEST_CODE_PERMISSION);

        // If any permission above not allowed by user, this condition will
          execute every time, else your else part will work
      }
    } catch (Exception e) {
      e.printStackTrace();
    }

    btnShowLocation = (Button) findViewById(R.id.button);

    // show location button click event
    btnShowLocation.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View arg0) {
        // create class object
        gps = new GPSTracker(MainActivity.this);

        // check if GPS enabled
        if(gps.canGetLocation()){

          double latitude = gps.getLatitude();
          double longitude = gps.getLongitude();

          // \n is for new line
          Toast.makeText(getApplicationContext(), "Your Location is - \nLat: "
            + latitude + "\nLong: " + longitude, Toast.LENGTH_LONG).show();
        }else{
          // can't get location
          // GPS or Network is not enabled
          // Ask user to enable GPS/network in settings
          gps.showSettingsAlert();
        }

      }
    });
  }}
```

Following is the content of the modified main activity file **GPSTracker.java**.

```
package com.example.tutorialspoint7.myapplication;
import          android.app.AlertDialog;import          android.app.Service;import
android.content.Context;import          android.content.DialogInterface;import
android.content.Intent;import                  android.location.Location;import
android.location.LocationListener;import
android.location.LocationManager;import                 android.os.Bundle;import
android.os.IBinder;import android.provider.Settings;import android.util.Log;
public class GPSTracker extends Service implements LocationListener {

  private final Context mContext;

  // flag for GPS status
  boolean isGPSEnabled = false;

  // flag for network status
  boolean isNetworkEnabled = false;

  // flag for GPS status
  boolean canGetLocation = false;

  Location location; // location
  double latitude; // latitude
  double longitude; // longitude

  // The minimum distance to change Updates in meters
  private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10; // 10
meters

  // The minimum time between updates in milliseconds
  private static final long MIN_TIME_BW_UPDATES = 1000 * 60 * 1; // 1 minute

  // Declaring a Location Manager
  protected LocationManager locationManager;

  public GPSTracker(Context context) {
    this.mContext = context;
    getLocation();
  }
  public Location getLocation() {
    try {
      locationManager                    =                    (LocationManager)
mContext.getSystemService(LOCATION_SERVICE);

      // getting GPS status
      isGPSEnabled                                                              =
locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);

      // getting network status
      isNetworkEnabled = locationManager
        .isProviderEnabled(LocationManager.NETWORK_PROVIDER);

      if (!isGPSEnabled && !isNetworkEnabled) {
        // no network provider is enabled
      } else {
        this.canGetLocation = true;
        // First get location from Network Provider
        if (isNetworkEnabled) {
          locationManager.requestLocationUpdates(
            LocationManager.NETWORK_PROVIDER,
            MIN_TIME_BW_UPDATES,
            MIN_DISTANCE_CHANGE_FOR_UPDATES, this);

          Log.d("Network", "Network");
          if (locationManager != null) {
            location = locationManager
              .getLastKnownLocation(LocationManager.NETWORK_PROVIDER);

            if (location != null) {
              latitude = location.getLatitude();
              longitude = location.getLongitude();
            }
          }
        }

        // if GPS Enabled get lat/long using GPS Services
        if (isGPSEnabled) {
          if (location == null) {
            locationManager.requestLocationUpdates(
              LocationManager.GPS_PROVIDER,
              MIN_TIME_BW_UPDATES,
              MIN_DISTANCE_CHANGE_FOR_UPDATES, this);

            Log.d("GPS Enabled", "GPS Enabled");
            if (locationManager != null) {
              location = locationManager
                .getLastKnownLocation(LocationManager.GPS_PROVIDER);

              if (location != null) {
                latitude = location.getLatitude();
                longitude = location.getLongitude();
              }
            }
          }
        }
      }

    } catch (Exception e) {
      e.printStackTrace();
    }

    return location;
  }
```

```
/**
  * Stop using GPS listener
  * Calling this function will stop using GPS in your app
* */

public void stopUsingGPS(){
  if(locationManager != null){
    locationManager.removeUpdates(GPSTracker.this);
  }
}

/**
  * Function to get latitude
* */

public double getLatitude(){
  if(location != null){
    latitude = location.getLatitude();
  }

  // return latitude
  return latitude;
}

/**
  * Function to get longitude
* */

public double getLongitude(){
  if(location != null){
    longitude = location.getLongitude();
  }
```

```
    // return longitude
    return longitude;
  }

  /**
    * Function to check GPS/wifi enabled
    * @return boolean
  * */

  public boolean canGetLocation() {
    return this.canGetLocation;
  }

  /**
    * Function to show settings alert dialog
    * On pressing Settings button will lauch Settings Options
  * */

  public void showSettingsAlert(){
    AlertDialog.Builder alertDialog = new AlertDialog.Builder(mContext);

    // Setting Dialog Title
    alertDialog.setTitle("GPS is settings");

    // Setting Dialog Message
    alertDialog.setMessage("GPS is not enabled. Do you want to go to settings
menu?");

    // On pressing Settings button
    alertDialog.setPositiveButton("Settings",                              new
DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog,int which) {
        Intent                          intent                          =                       new
Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
        mContext.startActivity(intent);
      }
    });

    // on pressing cancel button
    alertDialog.setNegativeButton("Cancel",                                                    new
DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int which) {
        dialog.cancel();
      }
    });

    // Showing Alert Message
    alertDialog.show();
  }

  @Override
  public void onLocationChanged(Location location) {
  }

  @Override
  public void onProviderDisabled(String provider) {
  }

  @Override
  public void onProviderEnabled(String provider) {
  }

  @Override
  public void onStatusChanged(String provider, int status, Bundle extras) {
  }
  @Override
  public IBinder onBind(Intent arg0) {
    return null;
  }}
```

Following will be the content of **res/layout/activity_main.xml** file –

```
<?xml version = "1.0" encoding = "utf-8"?><LinearLayout xmlns:android =
"http://schemas.android.com/apk/res/android"
  android:layout_width = "fill_parent"
  android:layout_height = "fill_parent"
  android:orientation = "vertical" >


  <Button
    android:id = "@+id/button"
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:text = "getlocation"/>
</LinearLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants –

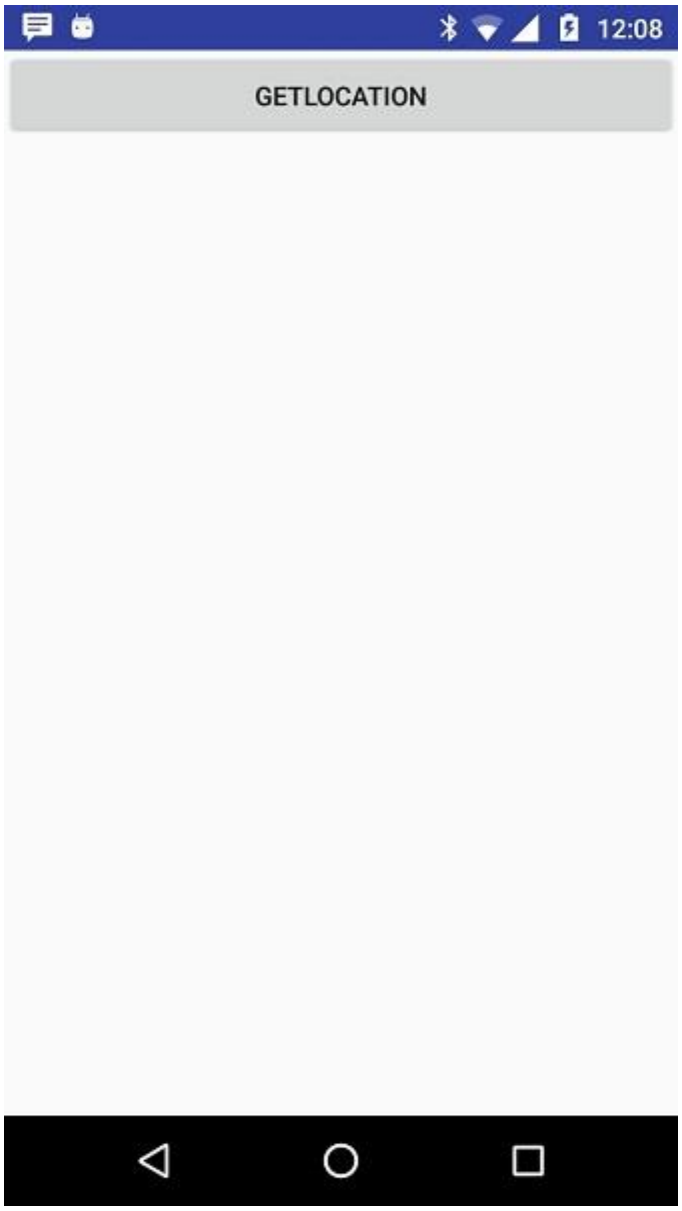
```
<?xml version = "1.0" encoding = "utf-8"?><resources>
  <string name = "app_name">Tutorialspoint</string></resources>
```

Following is the default content of **AndroidManifest.xml** –

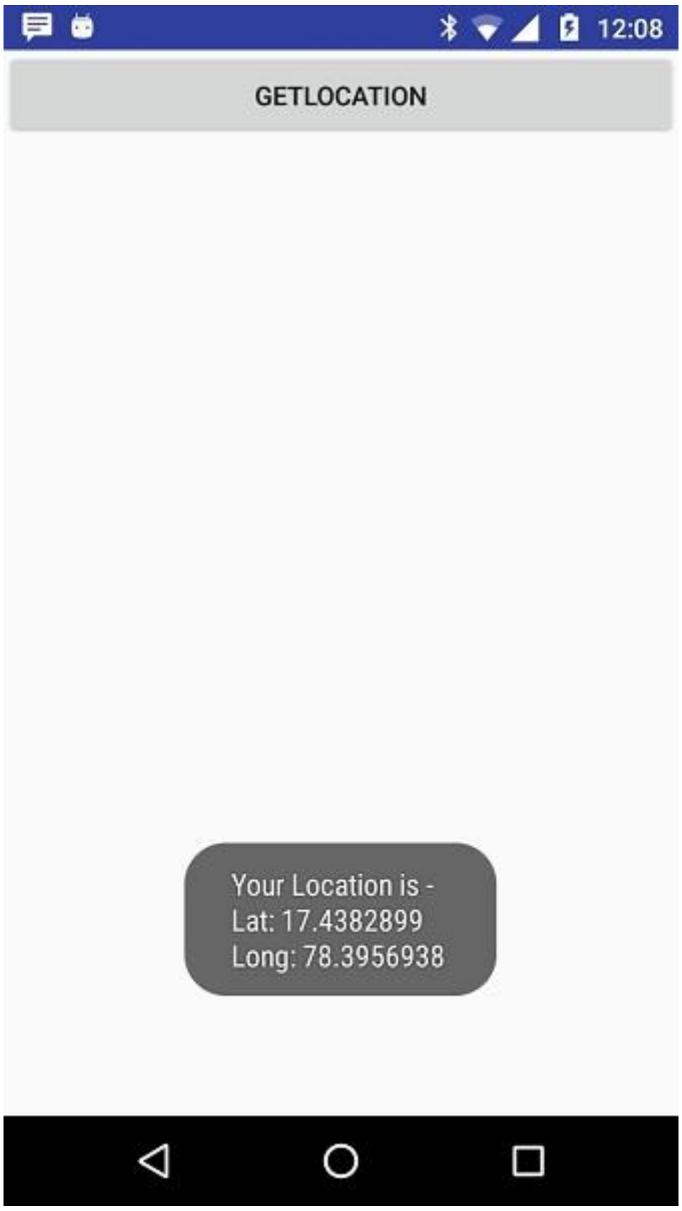
```
<?xml version = "1.0" encoding = "utf-8"?><manifest xmlns:android =
"http://schemas.android.com/apk/res/android"
  package = "com.example.tutorialspoint7.myapplication">
  <uses-permission android:name =
"android.permission.ACCESS_FINE_LOCATION" />
  <uses-permission android:name = "android.permission.INTERNET" />
  <application
```

```
    android:allowBackup = "true"
    android:icon = "@mipmap/ic_launcher"
    android:label = "@string/app_name"
    android:supportsRtl = "true"
    android:theme = "@style/AppTheme">

    <activity android:name = ".MainActivity">
      <intent-filter>
        <action android:name = "android.intent.action.MAIN" />

        <category android:name = "android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Let's try to run your **Tutorialspoint** application. I assume that, you have connected your actual Android Mobile device with your computer. To run the app from Android Studio, open one of your project's activity files and click Run ▶ icon from the toolbar. Before starting your application, Android studio installer will display following window to select an option where you want to run your Android application.



Now to see location select Get Location Button which will display location information as follows –



Your Location is -
Lat: 17.4382899
Long: 78.3956938