Introduction
A location-based service (LBS) is an information dissemination system that can be accessed by mobile devices through the mobile network. It is driven by the ability of the system to detect the geographical position of the mobile device.
Location Services

Introduction
Location Based Services are used in a variety of situations, such as
commercial,
entertainment,
emergency,
health,
work,
personal life, etc.

Examples:
• Locate the nearest bank, restaurant, gas station, hotel, golf course,
hospital, police station, etc.
• Provide transportation information on how to go from ‘here’ to ‘there’.
• Social networking is used to locate and reach events, friends and family
members.

How the Global Positioning System (GPS) Works?
The Global Positioning System (GPS) consists of 27 Earth-orbiting satellites (24 in operation and three extras in case one fails).

Developed by the USA as a military navigation system, but soon it opened to other civilian uses.

Each of these 3,000- to 4,000-pound solar-powered satellites circles the globe at about 12,000 miles (19,300 km), making two complete rotations every day.

The orbits are arranged so that at any time, anywhere on Earth, there are at least four satellites "visible" in the sky.

A GPS receiver’s job is to locate three or more of these satellites, figure out the distance to each, and use this information to deduce its own location. This operation is based on a mathematical principle called trilateration.

How the Global Positioning System (GPS) Works?

2-D Trilateration
Imagine you are somewhere in the United States and you are TOTALLY lost -- for whatever reason, you have absolutely no clue where you are. You find a friendly local and ask, "Where am I?" He says, "You are 625 miles from Boise, Idaho."

You ask somebody else where you are, and she says, "You are 690 miles from Minneapolis, Minnesota." Now you have two circles that intersect. You now know that you must be at one of these two intersection points.

If a third person tells you that you are 615 miles from Tucson, Arizona, you can eliminate one of the possibilities. You now know exactly where you are -- Denver, Colorado.

This same concept works in three-dimensional space, as well, but you’re dealing with spheres instead of circles.


3D-Trilateration
Rather than circles three spheres intersect to define your GPS receiver’s location.

For a visual explanation visit: http://electronics.howstuffworks.com/gadgets/travel/gps.htm

3D-Trilateration
Rather than circles three spheres intersect to define your GPS receiver’s location.

For a visual explanation visit: http://electronics.howstuffworks.com/gadgets/travel/gps.htm
Cell Tower Triangulation

An alternative method to determine the location of a cell phone is to estimate its distance to three nearby cell towers.

Distance of the phone to each antenna could be estimated based upon the lag time between the moment the tower sends a ping to the phone and receives the answering ping back.

Quite similar to the 2D-Trilateration Method.

Latitude & Longitude

Latitude in GPS-Decimal notation: +90.00000 (North) to -90.00000 (South)

Longitude GPS-Decimal notation: +180.000000 (East) to -180.000000 (West)
Android Location Classes

The Android API provides Location data based on a variety of methods including: **Cell Tower Triangulation**, and most commonly **GPS chip readings**.

*GPS is the most common location provider on the Android based phones.*

It offers the most accuracy. However the software can adapt to different hardware devices and their corresponding location data sources (providers).

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>A class representing an Address, i.e., a set of strings describing a location.</td>
</tr>
<tr>
<td>Criteria</td>
<td>A class indicating the application criteria for selecting a location provider.</td>
</tr>
<tr>
<td>Geocoder</td>
<td>A class for handling geocoding.</td>
</tr>
<tr>
<td>GpsSatellite</td>
<td>This class represents the current state of a GPS satellite.</td>
</tr>
<tr>
<td>GpsStatus</td>
<td>This class represents the current state of the GPS engine.</td>
</tr>
<tr>
<td>Location</td>
<td>A class representing a geographic location sensed at a particular time (a “fix”).</td>
</tr>
<tr>
<td>LocationManager</td>
<td>This class provides access to the system location services.</td>
</tr>
<tr>
<td>LocationProvider</td>
<td>An abstract superclass for location providers</td>
</tr>
</tbody>
</table>
Android Location Interfaces

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GpsStatus.Listener</td>
<td>Used for receiving notifications when GPS status has changed.</td>
</tr>
<tr>
<td>GpsStatus.NmeaListener</td>
<td>Used for receiving NMEA sentences from the GPS.</td>
</tr>
<tr>
<td>LocationListener</td>
<td>Used for receiving notifications from the LocationManager when the location has changed.</td>
</tr>
</tbody>
</table>

Location Class

- A class representing a geographic location sensed at a particular time (a "fix").
- A location consists of a latitude and longitude, a UTC timestamp and optionally information on altitude, speed, and bearing.
- Information specific to a particular provider or class of providers may be communicated to the application using `getExtras`, which returns a Bundle of key/value pairs.
- Each provider will only provide those entries for which information is available.

<table>
<thead>
<tr>
<th>Constants</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location FORMAT DEGREES</td>
<td>Constant used to specify formatting of a latitude or longitude in the form &quot;[+]DDD.DDDDD&quot; where D indicates degrees.</td>
</tr>
<tr>
<td>Location FORMAT MINUTES</td>
<td>Constant used to specify formatting of a latitude or longitude in the form &quot;[+]DDD:MM.MMMMM&quot; where D indicates degrees and M indicates minutes of arc (1 minute = 1/60th of a degree).</td>
</tr>
<tr>
<td>Location FORMAT SECONDS</td>
<td>Constant used to specify formatting of a latitude or longitude in the form &quot;[+]DDD:MM:SS.SSSSS&quot; where D indicates degrees, M indicates minutes of arc, and S indicates seconds of arc (1 minute = 1/60th of a degree, 1 second = 1/3600th of a degree).</td>
</tr>
</tbody>
</table>
Location Class – Useful Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void distanceBetween(double startLatitude, double startLongitude, double endLatitude, double endLongitude, float[] results)</td>
<td>Computes the approximate distance in meters between two locations, and optionally the initial and final bearings of the shortest path between them.</td>
</tr>
<tr>
<td>float getAccuracy()</td>
<td>Returns the accuracy of the fix in meters.</td>
</tr>
<tr>
<td>double getAltitude()</td>
<td>Returns the altitude of this fix.</td>
</tr>
<tr>
<td>float getBearing()</td>
<td>Returns the direction of travel in degrees East of true North.</td>
</tr>
<tr>
<td>Bundle getExtras()</td>
<td>Returns additional provider-specific information about the location fix as a Bundle.</td>
</tr>
<tr>
<td>double getLatitude()</td>
<td>Returns the latitude of this fix.</td>
</tr>
<tr>
<td>double getLongitude()</td>
<td>Returns the longitude of this fix.</td>
</tr>
<tr>
<td>String getProvider()</td>
<td>Returns the name of the provider that generated this fix, or null if it is not associated with a provider.</td>
</tr>
<tr>
<td>float getSpeed()</td>
<td>Returns the speed of the device over ground in meters/second.</td>
</tr>
<tr>
<td>long getTime()</td>
<td>Returns the UTC time of this fix, in milliseconds since January 1, 1970.</td>
</tr>
</tbody>
</table>

Location Manager

This class provides access to the system location services.

These services allow applications

1. To *obtain periodic updates of the device’s geographical location*,

2. or to fire an application-specified *Intent* when the *device enters the proximity of a given geographical location*.

You do not instantiate this class directly; instead, retrieve it through `Context.getSystemService(Context.LOCATION_SERVICE)`
**Location Manager – Useful Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void <code>addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent intent)</code></td>
<td>Sets a proximity alert for the location given by the position (latitude, longitude) and the given radius.</td>
</tr>
<tr>
<td>String <code>getBestProvider(Criteria criteria, boolean enabledOnly)</code></td>
<td>Returns the name of the provider that best meets the given criteria.</td>
</tr>
<tr>
<td>GpsStatus <code>getGpsStatus(GpsStatus status)</code></td>
<td>Retrieves information about the current status of the GPS engine.</td>
</tr>
<tr>
<td>Location <code>getLastKnownLocation(String provider)</code></td>
<td>Returns a Location indicating the data from the last known location fix obtained from the given provider.</td>
</tr>
<tr>
<td>LocationProvider <code>getProvider(String name)</code></td>
<td>Returns information associated with the location provider of the given name, or null if no provider exists by that name.</td>
</tr>
<tr>
<td>List&lt;String&gt; <code>getProviders(Criteria criteria, boolean enabledOnly)</code></td>
<td>Returns a list of the names of LocationProviders that satisfy the given criteria, or null if none do.</td>
</tr>
<tr>
<td>void <code>requestLocationUpdates(String provider, long minTime, float minDistance, PendingIntent intent)</code></td>
<td>Registers the current activity to be notified periodically by the named provider.</td>
</tr>
<tr>
<td>void <code>requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)</code></td>
<td>Registers the current activity to be notified periodically by the named provider.</td>
</tr>
<tr>
<td>void <code>setTestProviderStatus(String provider, int status, Bundle extras, long updateTime)</code></td>
<td>Sets mock status values for the given provider.</td>
</tr>
</tbody>
</table>

**LocationListener Class**

Used for receiving notifications from the `LocationManager` when the location has changed.

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

`requestLocationUpdates(Provider, minTime, minDistance, LocationListener)`
Location Services

LocationListener Class – Useful Methods

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract</td>
<td>onLocationChanged (Location location)</td>
<td>Called when the location has changed.</td>
</tr>
<tr>
<td>abstract</td>
<td>onProviderDisabled (String provider)</td>
<td>Called when the provider is disabled by the user.</td>
</tr>
<tr>
<td>abstract</td>
<td>onProviderEnabled (String provider)</td>
<td>Called when the provider is enabled by the user.</td>
</tr>
<tr>
<td>abstract</td>
<td>onStatusChanged (String provider, int status, Bundle extras)</td>
<td>Called when the provider status changes.</td>
</tr>
</tbody>
</table>

Public Methods

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract</td>
<td>getAccuracy()</td>
<td>Returns a constant describing horizontal accuracy of this provider.</td>
</tr>
<tr>
<td>String</td>
<td>getName()</td>
<td>Returns the name of this provider.</td>
</tr>
<tr>
<td>abstract</td>
<td>getPowerRequirement()</td>
<td>Returns the power requirement for this provider.</td>
</tr>
<tr>
<td>abstract</td>
<td>hasMonetaryCost()</td>
<td>true if the use of this provider may result in a monetary charge to the user, false if use is free.</td>
</tr>
<tr>
<td>boolean</td>
<td>meetsCriteria(Criteria criteria)</td>
<td>Returns true if this provider meets the given criteria, false otherwise.</td>
</tr>
<tr>
<td>abstract</td>
<td>requiresCell()</td>
<td>true if access to a cellular network (to make use of cell tower IDs) is needed, false otherwise.</td>
</tr>
<tr>
<td>abstract</td>
<td>requiresNetwork()</td>
<td>true if the provider requires access to a data network (e.g., the Internet), false otherwise.</td>
</tr>
<tr>
<td>abstract</td>
<td>requiresSatellite()</td>
<td>true if access to a satellite-based positioning system (e.g., GPS) is needed, false otherwise.</td>
</tr>
<tr>
<td>abstract</td>
<td>supportsAltitude()</td>
<td>Returns true if the provider is able to provide altitude information, false otherwise.</td>
</tr>
<tr>
<td>abstract</td>
<td>supportsBearing()</td>
<td>Returns true if the provider is able to provide bearing information, false otherwise.</td>
</tr>
<tr>
<td>abstract</td>
<td>supportsSpeed()</td>
<td>Returns true if the provider is able to provide speed information, false otherwise.</td>
</tr>
</tbody>
</table>

Constants:

- LocationProvider.AVAILABLE
- LocationProvider.OUT_OF_SERVICE
- LocationProvider.TEMPORARILY_UNAVAILABLE
Location Services

LocationProvider Class

An abstract superclass for location providers.

A location provider supplies periodic reports on the geographical location of the device.

Each provider has a set of criteria under which it may be used; for example, some providers require GPS hardware and visibility to a number of satellites; others require the use of the cellular radio, or access to a specific carrier’s network, or access to the internet.

They may also have different battery consumption characteristics or monetary costs to the user.

The Criteria class allows providers to be selected based on user-specified criteria.

Example – Obtain Location from GPS.

In this example we request GPS services and display latitude and longitude values on the UI. Additionally we deliver an SMS with this information.

Notes

1. Observe the GPS chip is not a synchronous device that will immediately respond to a “give me a GPS reading” call.

2. In order to engineer a good solution that takes into account the potential delays in obtaining location data we place the UI in the main activity and the request for location in a background service.

3. Remember the service runs in the same process space as the main activity, therefore for the sake of responsiveness we must place the logic for location data request in a separate parallel thread.

3. A thread (unlike an Activity) needs the presence of aLoopercontrol to manage IPC message sending. This implies and additionalLooper.prepareandLooper.loopmethods surrounding thelocationUpdatemethod.
Example – Obtain Location from GPS.

Use the DDMS > Emulator Control panel to enter test data reflecting Latitude and Longitude.

Select emulator 5554.

Press the ‘Send’ button to transmit the data.

A text message will be sent to a second emulator (5556).

Example – Obtain Location from GPS.

```xml
    <EditText android:id="@+id/txtMsg" android:layout_width="fill_parent" android:layout_height="120px" android:textSize="12sp" />
    <Button android:id="@+id/btnStopService" android:layout_width="151px" android:layout_height="wrap_content" android:text="Stop Service" />
</LinearLayout>
```
Example – Obtain Location from GPS.

Manifest

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cis493.mappinggps"
    android:versionCode="1"
    android:versionName="1.0">
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name"
        android:debuggable="true">
        <activity android:name="MyGPS"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <service android:name="MyGpsService"/>
    </application>
    <uses-sdk android:minSdkVersion="2"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
</manifest>
```

Main Activity: MyGPS

```java
public class MyGPS extends AppCompatActivity {

    // Request GPS location, show lat & long, deliver a text-message
    // Application logic and its BroadcastReceiver in the same class

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```
Location Services

Example – Obtain Location from GPS.

Main Activity: MyGPS

```java
public class MyGPS extends Activity {
    TextView txtMsg;
    Button btnStopService;
    ComponentName service;
    Intent intentMyService;
    BroadcastReceiver receiver;
    String GPS_FILTER = "cis493.action.GPS_LOCATION";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        txtMsg = (TextView) findViewById(R.id.txtMsg);
        intentMyService = new Intent(this, MyGpsService.class);
        service = startService(intentMyService);
        txtMsg.setText("MyGpsService started - (see DDMS Log)");
        // register & define filter for local listener
        IntentFilter mainFilter = new IntentFilter(GPS_FILTER);
        receiver = new MyMainLocalReceiver();
        registerReceiver(receiver, mainFilter);
    }

    public void onClick(View v) {
        try {
            stopService(new Intent(intentMyService));
            txtMsg.setText("After stopping Service: \n" + 
                           service.getClassName());
            btnStopService.setText("Finished");
            btnStopService.setClickable(false);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Main Activity:

```java
btnStopService = (Button) findViewById(R.id.btnStopService);
btnStopService.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        try {
            stopService(new Intent(intentMyService));
            txtMsg.setText("After stopping Service: \n" + 
                           service.getClassName());
            btnStopService.setText("Finished");
            btnStopService.setClickable(false);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
```
Location Services

Example – Obtain Location from GPS.

Main Activity: MyGPS

```java
@Override
protected void onDestroy() {
    super.onDestroy();
    try {
        stopService(intentMyService);
        unregisterReceiver(receiver);
    } catch (Exception e) {
        Log.e("MAIN-DESTROY>>>", e.getMessage());
    }

    Log.e("MAIN-DESTROY>>>", "Adios");
} // onDestroy
```

Example – Obtain Location from GPS.

Main Activity: MyGPS

```java
local RECEIVER
private class MyMainLocalLocalReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context localContext, Intent callerIntent) {
        double latitude = callerIntent.getDoubleExtra("latitude", -1);
        double longitude = callerIntent.getDoubleExtra("longitude", -1);

        Log.e("MAIN>>>", Double.toString(latitude));
        Log.e("MAIN>>>", Double.toString(longitude));

        String msg = " lat: " + Double.toString(latitude) + " lon: " + Double.toString(longitude);

        txtMsg.append("\n" + msg);

        //testing the SMS-texting feature
        texting(msg);
    }
} //MyMainLocalLocalReceiver
```
**Location Services**

**Example – Obtain Location from GPS.**

**Main Activity: MyGPS**

```java
private void texting(String msg) {
    try {
        SmsManager smsMgr = SmsManager.getDefault();
        // smsMgr.sendTextMessage(destinationAddress, senderAddress, text, sentIntent, deliveryIntent)
        //----------------------------------------------------------
        smsMgr.sendTextMessage("5556", "5551234", "Please meet me at: "+ msg, null, null);
    } catch (Exception e) {
        Toast.makeText(this, "texting\n" + e.getMessage(), 1).show();
    }
    // texting
} //MyGPS
```

**Example – Obtain Location from GPS.**

**Main Activity: MyGpsService**

```java
package cis493.mappinggps;
import android.app.Service;
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.os.Looper;
import android.util.Log;
import android.widget.Toast;

public class MyGpsService extends Service {
    String GPS_FILTER = "cis493.action.GPS_LOCATION";
    Thread triggerService;
    LocationManager lm;
    GPSListener myLocationListener;
    boolean isRunning = true;
```
Example – Obtain Location from GPS.

Main Activity: MyGpsService

```java
@Override
public IBinder onBind(Intent arg0) {
   return null;
}

@Override
public void onCreate() {
   super.onCreate();
}

@Override
public void onStart(Intent intent, int startId) {
   super.onStart(intent, startId);
   Log.e("<<MyGpsService-onsStart>>", "I am alive-GPS!");
   // we place the slow work of the service in its own thread so the
   // response we send our caller who run a "startService(...)" method
   // gets a quick OK from us.
```

triggerService = new Thread(new Runnable() {
   public void run() {
      try {
         // try to get your GPS location using the LOCATION.SERVIVE provider
         lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
         // This listener will catch and disseminate location updates
         myLocationListener = new GPSListener();
         long minTime = 10000; // frequency update: 10 seconds
         float minDistance = 0; // frequency update: 50 meter
         lm.requestLocationUpdates(LocationManager.GPS_PROVIDER,
         minTime, minDistance, myLocationListener);
        Looper.loop();
      } catch (Exception e) {
         e.printStackTrace();
      }
      // run
   }
});
triggerService.start();
```
Example – Obtain Location from GPS

Main Activity: MyGpsServive

```java
private class GPSListener implements LocationListener {
    public void onLocationChanged(Location location) {
        // capture location data sent by current provider
        double latitude = location.getLatitude();
        double longitude = location.getLongitude();
        // assemble data bundle to be broadcasted
        Intent myFilteredResponse = new Intent(GPS_FILTER);
        myFilteredResponse.putExtra("latitude", latitude);
        myFilteredResponse.putExtra("longitude", longitude);
        Log.e(">>GPS_Service<<", "Lat:" + latitude + " lon:" + longitude);
        // send the location data out
        sendBroadcast(myFilteredResponse);
    }

    public void onProviderDisabled(String provider) {
    }

    public void onProviderEnabled(String provider) {
    }

    public void onStatusChanged(String provider, int status, Bundle extras) {
    }
}; // GPSListener class
```

Part of the listener’s interface
JARGON:

Bearing
is the angle (East-ward) between a line connecting two points (source, destination) and a north-south line, or meridian.

NMEA (National Marine Electronics Association)
The NMEA 2000 standard contains the requirements for the minimum implementation of a serial-data communications network to interconnect marine electronic equipment onboard vessels. Equipment designed to this standard will have the ability to share data, including commands and status, with other compatible equipment over a single signaling channel.
Reference: http://www.nmea.org/content/nmea_standards/white_papers.asp

UTC - Coordinated Universal Time
Is a time standard based on International Atomic Time (TAI) with leap seconds added at irregular intervals to compensate for the Earth’s slowing rotation.

Keyhole Markup Language
Use Eclipse’s DDMS > Emulator Control > KML tab to provide location data to your emulator using a KML file.
Example: File my_location_data.kml contains the following set of placemarks

```xml
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Placemark>
    <name>Station 46027</name>
    <description>Off the coast of Lake Earl</description>
    <Point>
      <coordinates>-124.38,41.85,0</coordinates>
    </Point>
  </Placemark>
  <Placemark>
    <name>Station 46020</name>
    <description>Outside the Golden Gate</description>
    <Point>
      <coordinates>-122.83,37.75,0</coordinates>
    </Point>
  </Placemark>
  <Placemark>
    <name>Station 46222</name>
    <description>San Pedro Channel</description>
    <Point>
      <coordinates>-118.31,33.61,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

Example taken from: