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.