**STEP BY STEP DIRECTIONS ON GOOGLE MAPS**

Direction between two location co-ordinates can be obtained using Directions API of google. It is assumed that you have already successfully integrated Google Maps with the application before reading this document.

1. **DIRECTIONS API:**

Official documentation for Directions API can be found at:

<https://developers.google.com/maps/documentation/directions/>

**USAGE LIMITS:**

There are some usage limits on Direction API:

1. 2,500 directions requests per 24 hour period.
2. When the mode of transportation is driving, walking, or cycling, each directions search counts as a single request.
3. Searching for transit directions counts as 4 requests.
4. Individual requests for driving, walking, or cycling directions may contain up to 8 intermediate waypoints in the request. Waypoints cannot be specified for transit requests.

[Google Maps API for Business](https://developers.google.com/maps/documentation/business) customers have higher limits:

1. 100,000 directions requests per 24 hour period.
2. 23 waypoints allowed in each request. Waypoints are not available for transit directions.

**DIRECTIONS REQUEST URL:**

The directions request takes the following form:

http://maps.googleapis.com/maps/api/directions/***output***?*parameters*

Directions can also be accessed over HTTPS using:

https://maps.googleapis.com/maps/api/directions/***outpu***t?*parameters*

where ***output*** can be one of the following:

* json: indicates response in JSON format.
* xml: indicates response as XML.

**Request Parameters:**

Some of the parameters in direction API are required while others are optional as listed below:

* **Required Parameters:**
* origin -

The address or textual latitude/longitude value from which you wish to calculate directions. If you pass an address as a string, the Directions service will geocode the string and convert it to a latitude/longitude coordinate to calculate directions. If you pass coordinates, ensure that no space exists between the latitude and longitude values.

* destination –

The address or textual latitude/longitude value of destination to which you wish to calculate directions. If you pass an address as a string, the Directions service will geocode the string and convert it to a latitude/longitude coordinate to calculate directions. If you pass coordinates, ensure that no space exists between the latitude and longitude values.

* sensor –

Indicates whether or not the directions request comes from a device with a location sensor. This value must be either ***true*** or ***false***.

Maps API for Business users must include valid client and signature parameters with their Directions requests. More details can be found at : [Maps API for Business Web Services](https://developers.google.com/maps/documentation/business/webservices).

* **Optional Parameters:**
* mode–

Specifies the mode of transport to use when calculating directions. The possible modes are:

1. driving (default): indicates standard driving directions using the road network.
2. walking:requests walking directions via pedestrian paths & sidewalks (where available).
3. bicycling: requests bicycling directions via bicycle paths & preferred streets (where available).
4. transit: requests directions via public transit routes (where available).

If you set the mode to "transit" you must also specify either a departure\_time or an arrival\_time.

* waypoints –

Specifies an array of waypoints. Waypoints alter a route by routing it through the specified location(s). A waypoint is specified as either a latitude/longitude coordinate or as an address which will be geocoded. Waypoints are not supported by the transit mode.

* alternatives –

It can either be set to true or false. If set to true, the directions API may return more than one route between origin and direction in response. Setting it to true may increase the response time.

* avoid –

Specifies that the route returned in response must avoid indicated features. This parameter supports two arguments:

1. tolls: indicates route returned must avoid tolls
2. highways: indicates route returned must avoid highways

Other optional parameters are language, units, region, arrival\_time and departure\_time. More information about these parameters can be found in above link.

**DIRECTIONS RESPONSE:**

The response may be in JSON or XML format depending upon the ***output*** flag in the directions request URL. JSON format is mostly preferred as it is light weight and easy to parse.

Consider following sample JSON response taken from [https://developers.google.com/maps/documentation/directions/#DirectionsResponses](https://developers.google.com/maps/documentation/directions/%23DirectionsResponses). This link also nicely explains all the elements in the JSON response.

{  
  "status": "OK",  
  "routes": [ {  
    "summary": "I-40 W",  
    "legs": [ {  
      "steps": [ {  
        "travel\_mode": "DRIVING",  
        "start\_location": {  
          "lat": 41.8507300,  
          "lng": -87.6512600  
        },  
        "end\_location": {  
          "lat": 41.8525800,  
          "lng": -87.6514100  
        },  
        "polyline": {  
          "points": "a~l~Fjk~uOwHJy@P"  
        },  
        "duration": {  
          "value": 19,  
          "text": "1 min"  
        },  
        "html\_instructions": "Head \u003cb\u003enorth\u003c/b\u003e on \u003cb\u003eS Morgan St\u003c/b\u003e toward \u003cb\u003eW Cermak Rd\u003c/b\u003e",  
        "distance": {  
          "value": 207,  
          "text": "0.1 mi"  
        }  
      },  
      ...  
      ... additional steps of this leg  
    ...  
    ... additional legs of this route  
      "duration": {  
        "value": 74384,  
        "text": "20 hours 40 mins"  
      },  
      "distance": {  
        "value": 2137146,  
        "text": "1,328 mi"  
      },  
      "start\_location": {  
        "lat": 35.4675602,  
        "lng": -97.5164276  
      },  
      "end\_location": {  
        "lat": 34.0522342,  
        "lng": -118.2436849  
      },  
      "start\_address": "Oklahoma City, OK, USA",  
      "end\_address": "Los Angeles, CA, USA"  
    } ],

"copyrights": "Map data ©2010 Google, Sanborn",

"overview\_polyline": {

"points": "a~l~Fjk~uOnzh@vlbBtc~@tsE`vnApw{A`dw@~w\\|tNtqf@l{Yd\_Fblh@rxo@b}@xxSfytAblk@xxaBeJxlcBb~t@zbh@jc|Bx}C`rv@rw|@rlhA~dVzeo@vrSnc}Axf]fjz@xfFbw~@dz{A~d{A|zOxbrBbdUvpo@`cFp~xBc`Hk@nurDznmFfwMbwz@bbl@lq~@loPpxq@bw\_@v|{CbtY~jGqeMb{iF|n\\~mbDzeVh\_Wr|Efc\\x`Ij{kE}mAb~uF{cNd}xBjp]fulBiwJpgg@|kHntyArpb@bijCk\_Kv~eGyqTj\_|@`uV`k|DcsNdwxAott@r}q@\_gc@nu`CnvHx`k@dse@j|p@zpiAp|gEicy@`omFvaErfo@igQxnlApqGze~AsyRzrjAb\_\_@ftyB}pIlo\_BflmA~yQftNboWzoAlzp@mz`@|}\_@fda@jakEitAn{fB\_a]lexClshBtmqAdmY\_hLxiZd~XtaBndgC"

},

"warnings": [ ],

"waypoint\_order": [ 0, 1 ],

"bounds": {

"southwest": {

"lat": 34.0523600,

"lng": -118.2435600

},

"northeast": {

"lat": 41.8781100,

"lng": -87.6297900

}

}

} ]

}

In order to show the path on the map and list of step by step directions, the JSON response needs to be parsed.

1. **CODE SNIPPETS FOR MAKING DIRECTIONS API CALL, PARSING RESPONSE, SHOWING THE PATH AND STEP BY STEP DIRECTIONS ON MAP:**

Let us consider a scenario where we have displayed a google map with two markers on it in an android activity. One of the markers is at the origin location and other at destination and we have to obtain direction from origin to destination on click of say ‘Get Directions’ button which is in the same layout.

Therefore, we need to set onClickListener for ‘Get Directions’ button and override onClick() method in the following manner:

public void onClick(View v) {

if (v.getId() == R.id.*btn\_get\_directions*) {

getDirections(mSource, mDestination);

}

}

where mSource and mDestination are member fields containing the source/origin and destination location respectively. They are instance of LatLng class present in com.google.android.gms.maps.model.\* package.

Now, in getDirections() method we build the URL and execute an Asynctask which will make directions API call:

private void getDirections(LatLng source, LatLng destination)

{

String urlParameters = "origin="+source.latitude+","+source.longitude+"&destination="+destination.latitude+","+destination.longitude+"&mode=”+mDestinationMode+”&sensor=false&alternatives=false";

String url = “http://maps.googleapis.com/maps/api/directions/json?”+urlParameters;

// Check if there is internet connection. If yes, execute asynctask otherwise ask user to //check internet connection

new FetchDirectionTask().execute(url);

}

Above method just builds the directions API request URL with appropriate parameters and uses this URL for executing a AsyncTask which will make actual API call in background. The parameters of URL in above snippet are not encoded. It is always a good practice to encode them.

mDestinationMode here is one of the driving, walking or bicycling modes selected by user.

public class FetchDirectionTask extends AsyncTask<String,String,String>

{

@Override

protected void onPreExecute() {

super.onPreExecute();

//display a progress dialog

}

@Override

protected String doInBackground(String... params) {

String api\_url=params[0];

try

{

URL url=new URL(api\_url);

Log.d("Service URL", api\_url);

HttpURLConnection conn=(HttpURLConnection)url.openConnection();

conn.setDoInput(true);

conn.setConnectTimeout(10000);

conn.setRequestProperty("Accept-Charset", "UTF-8");

conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8");

conn.setRequestProperty("User-Agent","Mozilla/5.0 ( compatible ) ");

//start listening to the stream

BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));

String line;

StringBuffer response=new StringBuffer();

while((line=in.readLine())!=null)

{

response.append(line);

}

in.close();

}

catch(java.net.SocketTimeoutException ex)

{

Log.e("Exception in URL request", ex.toString());

response = new StringBuffer(mContext.getResources().getString(R.string.connectionTimeOut));

return response.toString();

}

catch(Exception e)

{

Log.e("Exception in URL request", e.toString());

response = new StringBuffer("Exception in url request");

return response.toString();

}

finally

{

conn.disconnect();

}

return response.toString();

}

protected void onPostExecute(String result)

{

super.onPostExecute(result);

//dismiss progress dialog

if(result.equals(getString(R.string.connectionTimeOut)))

{

Toast.makeText(MapDirectionsActivity.this, "Connection Timed Out", Toast.LENGTH\_SHORT).show();

}

else

{

parseDirectionData(result);

}

}

}

In overridden doInBackground() method of above class which extends Asynctask, the direction API call is made and response received is passed to parseDirectionData() method from onPostExecute().

parseDirectionData() method parses the JSON directions API response received and calls appropriate methods for displaying path and step by step directions in Map.

Before defining parseDirectionData() method, we will create a class called Step for storing details of each step in the route during parsing. This class contains some member fields and corresponding getter and setter methods.

Step.java:

import java.io.Serializable;

import com.google.android.gms.maps.model.LatLng;

public class Step implements Serializable{

double startLocationLatitiude;

double startLocationLongitude;

double endLocationLatitude;

double endLocationLongitude;

String instruction;

String distance;

String duration;

/\*

Getter and setter methods for all above member variables

\*/

}

parseDirectionData():

public void parseDirectionData(String result) {

// TODO Auto-generated method stub

try {

JSONObject jsonResult = new JSONObject(result);

if(jsonResult.getString("status").equalsIgnoreCase("OK"))

{

JSONArray arrayRoutes = jsonResult.getJSONArray("routes");

for(int i = 0 ; i < arrayRoutes.length(); i++)

{

JSONObject jsonRoute = arrayRoutes.getJSONObject(i);

JSONArray arrayLegs = jsonRoute.getJSONArray("legs");

for(int j = 0; j < arrayLegs.length(); j++)

{

JSONObject jsonLeg = arrayLegs.getJSONObject(j);

JSONObject jsonDistance = jsonLeg.getJSONObject("distance");

mTotalDistance = jsonDistance.getString("text");

JSONObject jsonDuration = jsonLeg.getJSONObject("duration");

mTotalDuration = jsonDuration.getString("text");

//Getting the steps of the route and storing them into an ArrayList //for adding marker and showing corresponding instruction on map for steps JSONArray arraySteps = jsonLeg.getJSONArray("steps");

mStepList = new ArrayList<Step>(arraySteps.length());

for(int k = 0; k < arraySteps.length(); k++)

{

JSONObject jsonStep = arraySteps.getJSONObject(k);

JSONObject jsonStartAddress = jsonStep.getJSONObject("start\_location");

JSONObject jsonEndAddress = jsonStep.getJSONObject("end\_location");

JSONObject jsonStepDistance = jsonStep.getJSONObject("distance");

JSONObject jsonStepDuration = jsonStep.getJSONObject("duration");

String html\_instruction = jsonStep.getString("html\_instructions");

LatLng startLocation = new LatLng(jsonStartAddress.getDouble("lat"), jsonStartAddress.getDouble("lng"));

LatLng endLocation = new LatLng(jsonEndAddress.getDouble("lat"), jsonEndAddress.getDouble("lng"));

String stepDistance = jsonStepDistance.getString("text");

String stepDuration = jsonStepDuration.getString("text");

Step step = new Step();

step.setStartLocationLatitiude(startLocation.latitude);

step.setStartLocationLongitude(startLocation.longitude);

step.setEndLocationLatitude(endLocation.latitude);

step.setEndLocationLongitude(endLocation.longitude);

step.setInstruction(html\_instruction);

step.setDistance(stepDistance);

step.setDuration(stepDuration);

mStepList.add(step);

}

showStepByStepDirection(mStepList);

}

JSONObject jsonOverviewPolyline = jsonRoute.getJSONObject("overview\_polyline");

String encodedPoints = jsonOverviewPolyline.getString("points");

drawPath(encodedPoints);

}

}

else

{

Toast.makeText(this, mDirectionMode + " directions unavailable. Please select some other mode.", Toast.LENGTH\_SHORT).show();

Log.e("Direction API returned", "Zero result");

}

} catch (JSONException e) {

// TODO Auto-generated catch block

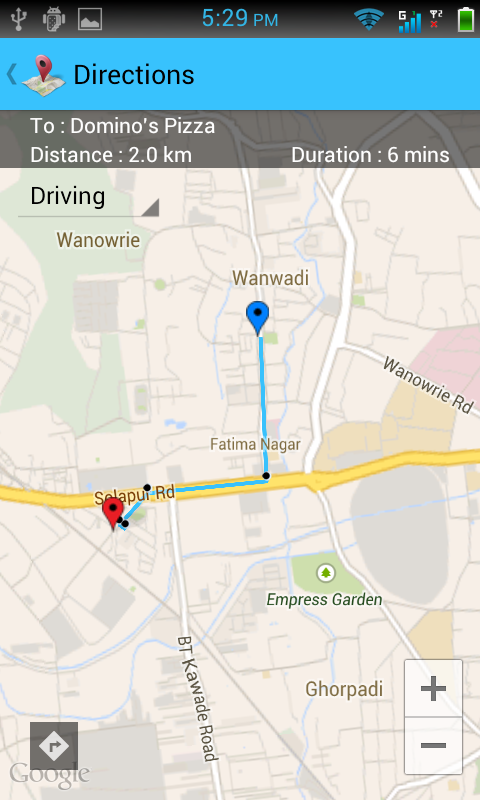
e.printStackTrace();

}

}

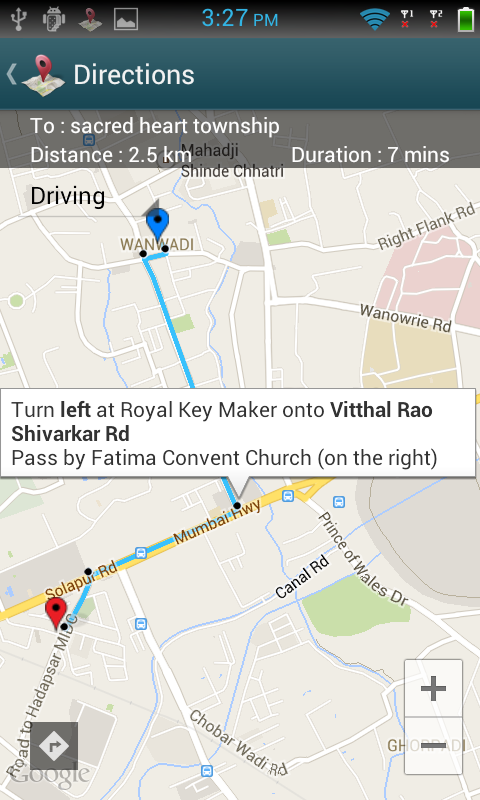
The JSON array “route” within returned JSON contains a JSON object called “overview\_polyline”. This JSON object contains an element having key “points” and its value is an encoded string. This encoded string on decoding returns a set of points (locations on map) through which polyline is to be drawn. parseDirectionData calls a method called drawPath() and passes this encoded string to it. drawPath() method decodes the string and draws the path from source to destination on the map.

parseDirectionData() calls one more method called showStepByStepDirection() which adds marker at the starting location of every step and shows corresponding instruction on click of that marker.



**drawPath() method will draw the blue line on the map** showing path between source and destination.

**showStepByStepDirection()** method will draw block points (which are nothing but marker) on the map and **show travelling instruction whenever a point is clicked** as shown in the screenshot below:



drawPath():

private void drawPath(String encodedPoints) {

List<LatLng> points = decodePoly(encodedPoints);

PolylineOptions options = new PolylineOptions().add(points.get(0));

for(int i = 1; i < points.size(); i++)

{

options.add(points.get(i)).width(4).color(Color.*parseColor*("#38C2FD")).geodesic(true);

}

mLine = mMap.addPolyline(options);

}

Now we will define decodePoly() method which will decode the encoded “points” string returned in JSON response.

private List<LatLng> decodePoly(String encodedPoints) {

List<LatLng> poly = new ArrayList<LatLng>();

int index = 0, len = encodedPoints.length();

int lat = 0, lng = 0;

while (index < len) {

int b, shift = 0, result = 0;

do {

b = encodedPoints.charAt(index++) - 63;

result |= (b & 0x1f) << shift;

shift += 5;

} while (b >= 0x20);

int dlat = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));

lat += dlat;

shift = 0;

result = 0;

do {

b = encodedPoints.charAt(index++) - 63;

result |= (b & 0x1f) << shift;

shift += 5;

} while (b >= 0x20);

int dlng = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));

lng += dlng;

LatLng p = new LatLng( (((double) lat / 1E5)),

(((double) lng / 1E5) ));

poly.add(p);

}

return poly;

}

Now, we will define showStepByStepDirections() method for adding markers on the path indicating a step and showing corresponding instructions in infoWindow.

private void showStepByStepDirection(ArrayList<Step> stepList)

{

Marker[] stepMarkers = new Marker[stepList.size()];

for(int i = 0; i < stepList.size(); i++)

{

Step step = stepList.get(i);

LatLng startLocation = new LatLng(step.getStartLocationLatitiude(), step.getStartLocationLongitude());

stepMarkers[i] = mMap.addMarker(new MarkerOptions()

.position(startLocation)

.icon(BitmapDescriptorFactory.*fromResource*(R.drawable.*step*))

.snippet(step.getInstruction()));

}

}

This method takes only one parameter ArrayList of type Step. This ArrayList is populated and passed to this method from parseDirectionData() method.

That’s it, both path and step by step directions will be visible on map now.

We can show this step by step directions on a separate listView also using the data stored in arrayList(which was obtained by parsing JSON).