

## ATLS 4120/5120: Mobile Application Development

### Week 14: Google Location

#### Accessing Google APIs

When you want to make a connection to one of the Google APIs provided in the Google Play services library you need to create an instance of `GoogleApiClient`. The Google API Client provides a common entry point to all the Google Play services and manages the network connection between the user's device and each Google service.

<https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient>

`GoogleApiClient.Builder` is a class to configure a `GoogleApiClient`.

<https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder>

There are two interfaces -- `GoogleApiClient.ConnectionCallbacks` and `GoogleApiClient.OnConnectionFailedListener`

#### Location Updates

Along with displaying a map you can also get the user's location, get location updates, and display locations on the map.

<https://developer.android.com/training/location/receive-location-updates.html>

#### Permissions

<https://developer.android.com/training/permissions/requesting.html>

Android's permission system has been one of the biggest security concerns all along since those permissions are asked for at install time. Once installed, the application will be able to access all of things granted without any user's acknowledgement what exactly application does with the permission. In Android 6.0 Marshmallow(API 23) the permission system was redesigned and apps are not granted any permission at installation time. Instead, apps have to ask user for a permission one-by-one at runtime.

This permission request dialog will *not* launch automatically, you must call it programmatically. In the case that an app tries to call some function that requires a permission which user has not granted yet, the function will suddenly throw an exception which will lead to the application crashing.

Also, users are able to revoke the granted permission anytime through a device's settings.

Older apps running pre-API23 will use the old behavior.

#### **Map**

Adding on to the Map app.

In your app go into Gradle Scripts and open `build.gradle(Module: app)` and under dependencies add the following with the same version you used for maps.

implementation '**com.google.android.gms:play-services-location:11.6.0**'

In order to access the user's location we need to implement the following callbacks in our class.

1. **`GoogleApiClient.ConnectionCallbacks`**: This callback will have a public function `onConnected()` which will be called whenever device is connected and disconnected.

2. **GoogleApiClient.OnConnectionFailedListener**: Provides callbacks for scenarios that result in a failed attempt to connect the client to the service. Whenever connection is failed `onConnectionFailed()` will be called.
3. **LocationListener**: This callback will be called whenever there is change in location of device. Function `onLocationChanged()` will be called.

Add these in the “implement” section after `OnMapReadyCallback`.

```
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback,  
GoogleApiClient.ConnectionCallbacks, GoogleApiClient.OnConnectionFailedListener,  
LocationListener
```

As soon as you add these you will get the red squiggle lines under the callbacks because we haven’t implemented their required methods.

Move your mouse over them and click `control+Enter` (alt for Windows) and select `Implement Methods`. You will see 4 selected (`onConnected()`, `onConnectionSuspended()`, `onConnectionFailed()` and `onLocationChanged()`) select OK and these will be added for you and the red squiggles should go away.

### Google Play Services

Now we need to initialize Google Play Services.

Add a `GoogleApiClient` object at the top of your class.

*// The entry point to Google Play services, used by the Places API and Fused Location Provider.*

```
private GoogleApiClient mGoogleApiClient;
```

We’re going to create a builder method `buildGoogleApiClient()` to initialize Google Play services.

```
private synchronized void buildGoogleApiClient() {  
    //The GoogleApiClient class is the main entry point for integrating with Google Play Services  
    mGoogleApiClient = new GoogleApiClient.Builder(this)  
        .addConnectionCallbacks(this) //connection callbacks are called when the client is connected  
        or disconnected  
        .addOnConnectionFailedListener(this) //handles failed attempt of connect client to service  
        .addApi(LocationServices.API) //adds the Google Play Service LocationServices API  
        .build(); //builds the GoogleApiClient object  
    mGoogleApiClient.connect(); //connect client  
}
```

And we’re going to call that method at the end of `onCreate()`

```
buildGoogleApiClient();
```

Now let’s update `onMapReady()`

*@Override*

```
public void onMapReady(GoogleMap googleMap) {  
    mMap = googleMap;  
  
    //set map type  
    mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
```

*//check for permission*

```

if (ContextCompat.checkSelfPermission(this.getApplicationContext(),
android.Manifest.permission.ACCESS_FINE_LOCATION)
    == PackageManager.PERMISSION_GRANTED) {
    //enables location layer
    mMap.setMyLocationEnabled(true);
}
}

```

### Fused Location Provider

<https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest>

Objects of the LocationRequest class are used to request location updates from the FusedLocationProviderApi which is what we'll use to access location information.

The Fused Location Provider analyses GPS, Cellular and Wi-Fi network location data in order to provide the highest accuracy data. It uses different device sensors to define if a user is walking, riding a bicycle, driving a car or just standing in order to adjust the frequency of location updates.

Add a LocationRequest object at the top of your class.

*// A request object to store parameters for requests to the FusedLocationProviderApi*

```

private LocationRequest mLocationRequest;

```

We will use this to get the last updated location in onConnected()

**@Override**

```

public void onConnected(@Nullable Bundle bundle) {
    mLocationRequest = new LocationRequest();
    mLocationRequest.setInterval(1000); //set the desired interval for active location updates, in
milliseconds
    mLocationRequest.setFastestInterval(1000); //set the fastest interval for location updates, in
milliseconds
    mLocationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY);
    //set priority of the request

    //check permission
    if (ContextCompat.checkSelfPermission(this.getApplicationContext(),
android.Manifest.permission.ACCESS_FINE_LOCATION)
        == PackageManager.PERMISSION_GRANTED) {
        //request location updates
        LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
mLocationRequest, this);
    }
}

```

Note: Although you'll see a line through FusedLocationApi and it says it's deprecated, I found the following so I have kept it.

Please continue using the FusedLocationProviderApi class and don't migrate to the FusedLocationProviderClient class until Google Play services version 12.0.0 is available, which is expected to ship in early 2018. Using the FusedLocationProviderClient before version 12.0.0 causes the client app to crash when Google Play services is updated on the device. We apologize for any inconvenience this may have caused.

## Location Changes

We've connected our app with APIs to get regular location intervals. The next step is to set up what happens when the user location changes. For that the Google LocationListener interface has a predefined function `onLocationChanged` that is called whenever the user location changes.

Add a Location and Marker object at the top of your class and then implement `onLocationChanged()`

*// location where the device is currently located*

**private** Location **mCurrentLocation**;

*// marker for current location*

**private** Marker **mCurrentLocationMarker**;

**@Override**

**public void** `onLocationChanged`(Location location) {

*//set current location to the new location*

**mCurrentLocation** = location;

*//define an object of the Google LatLng class with location coordinates*

LatLng latLng = **new** LatLng(location.getLatitude(), location.getLongitude());

*//check to see if there's a current marker*

**if** (**mCurrentLocationMarker** == **null**) {

*//define an object of the Google MarkerOptions class*

MarkerOptions markerOptions = **new** MarkerOptions();

markerOptions.position(latLng);

markerOptions.title("**Current Position**");

*//place current location marker*

**mCurrentLocationMarker** = **mMap**.addMarker(markerOptions);

} **else**{

*//set position of existing marker*

**mCurrentLocationMarker**.setPosition(latLng);

}

*//move map camera*

**mMap**.moveCamera(CameraUpdateFactory.newLatLng(latLng));

**mMap**.animateCamera(CameraUpdateFactory.zoomTo(11));

}

The marker can be configured as well as set up to be dragged or clicked.

<https://developers.google.com/android/reference/com/google/android/gms/maps/model/Marker>

You can have as many markers as you want. Remove them by calling `remove()` or clear the whole map with `clear()`

You can stop monitoring for location changes by calling `removeLocationUpdates()`

## Permissions

In Android 6.0 Marshmallow(API 23) the app will not be granted any permission at installation time. Instead, the app has to ask user for permissions at runtime.

Add a constant int for our location permission

*//can be any value >0*

```
private static final int MY_PERMISSIONS_REQUEST_LOCATION = 1;
```

Create the method to check permissions.

```
public void checkLocationPermission(){
```

```
    //check for permission
```

```
    if (ContextCompat.checkSelfPermission(this.getApplicationContext(),  
    android.Manifest.permission.ACCESS_FINE_LOCATION) !=  
    PackageManager.PERMISSION_GRANTED) {
```

```
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,  
        android.Manifest.permission.ACCESS_FINE_LOCATION)) {
```

```
            // returns true if the app has requested this permission previously and the user denied the request
```

```
                // Show an explanation to the user *asynchronously* -- don't block
```

```
                // this thread waiting for the user's response! After the user
```

```
                // sees the explanation, try again to request the permission.
```

```
                //request permission once explanation has been shown
```

```
                ActivityCompat.requestPermissions(this, new  
                String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},  
                MY_PERMISSIONS_REQUEST_LOCATION);
```

```
            // returns false if user has chosen Don't ask again option when it previously asked for permission
```

```
        } else {
```

```
            //no explanation needed, request permissions
```

```
            ActivityCompat.requestPermissions(this, new  
            String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},  
            MY_PERMISSIONS_REQUEST_LOCATION);
```

```
        }
```

```
    }
```

```
}
```

And call it in onCreate()

```
checkLocationPermission();
```

onRequestPermissionsResult() is invoked for every call on requestPermissions()

**@Override**

```
public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) {  
    switch (requestCode) {
```

```
        case MY_PERMISSIONS_REQUEST_LOCATION: {
```

```
            // If request is cancelled the result arrays are empty
```

```
            if (grantResults.length > 0 && grantResults[0] ==  
            PackageManager.PERMISSION_GRANTED) {
```

```
                // Permission was granted
```

```
                if (ContextCompat.checkSelfPermission(this,  
                android.Manifest.permission.ACCESS_FINE_LOCATION) ==
```

```

PackageManager.PERMISSION_GRANTED) {
    if (mGoogleApiClient == null) {
        buildGoogleApiClient();
    }
    mMap.setMyLocationEnabled(true);
    Toast.makeText(this, "permission granted", Toast.LENGTH_LONG).show();
}
} else { // Permission denied
    Toast.makeText(this, "permission denied", Toast.LENGTH_LONG).show();
}
}
// add other 'case' lines to check for other permissions your app might request
}
}

```

In your AndroidManifest file comment the following line to see permissions denied Toast.  
`<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />`

Remember that requestPermissions() will only be called in Android 6 or later.

Sources:

<http://www.androidtutorialpoint.com/intermediate/android-map-app-showing-current-location-android/>  
<https://developers.google.com/maps/documentation/android-api/current-place-tutorial> (save map state at end)