Request permission to access nearby Wi-Fi devices

bookmark_border Apps that target Android 13 (API level 33) or higher and manage Wi-Fi connections must request the NEARBY_WIFI_DEVICES runtime permission. This permission makes it easier to justify an app's access of nearby Wi-Fi devices; on previous versions of Android, these apps needed to declare the ACCESS_FINE_LOCATION permission instead.

Caution: If your app tries to call a Wi-Fi API without the proper permission, a **SecurityException** occurs.

Permission is part of the nearby devices group

The NEARBY_WIFI_DEVICES permission is part of the **Nearby devices** permission group. This group, added in Android 12 (API level 31), also includes permissions related to Bluetooth and Ultra-wideband. When you request any combination of permissions from this permission group, the system shows a single runtime dialog and asks the user to approve your app's access to nearby devices. In system settings, the user must enable and disable the **Nearby devices** permissions as a group; for example, users can't disable Wi-Fi access but keep Bluetooth access enabled for a given app.

Strongly assert that your app doesn't derive physical location

When you target Android 13 or higher, consider whether your app ever derives location information from Wi-Fi APIs; if not, you should strongly assert that. To make this assertion, set the usesPermissionFlags attribute to neverForLocation in your app's manifest file, as shown in the following code snippet. This process is similar to the one you do when you assert that Bluetooth device information is never used for location:

```
<manifest ...>
    <uses-permission
        android:name="android.permission.NEARBY_WIFI_DEVICES"
        android:usesPermissionFlags="neverForLocation" />
```

```
  <application ...>
    ...
  </application>
</manifest>
```

Previous versions and some APIs require location permission

Several Wi-Fi APIs require the ACCESS_FINE_LOCATION permission, even when your app targets Android 13 or higher. Examples include the following methods from the WifiManager class:

- getScanResults()
- startScan()

Also, because the NEARBY_WIFI_DEVICES permission is available only on Android 13 and higher, you should keep any declarations for ACCESS_FINE_LOCATION to provide backward compatibility in your app. However, as long as your app doesn't otherwise rely on precise location information, you can set the maximum SDK version of this permission to 32, as shown in the following code snippet:

```
<manifest ...>
                                                    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"
          android:maxSdkVersion="32" />
  <application ...>
    ...
  </application>
</manifest>
```

Check for APIs that require the permission

If your app targets Android 13 or higher, you must declare the NEARBY_WIFI_DEVICES permission to call any of the following Wi-Fi APIs:

- WifiManager

- startLocalOnlyHotspot()
- WifiAwareManager
- attach()
- WifiAwareSession
- publish()
- subscribe()
- WifiP2pManager
- addLocalService()
- connect()
- createGroup()
- discoverPeers()
- discoverServices()
- requestDeviceInfo()
- requestGroupInfo()
- requestPeers()
- WifiRttManager
- startRanging()

Wi-Fi access workflows

Figure 1 shows the Wi-Fi access workflow on devices that run Android 13 or higher, for apps that target Android 13 or higher. Note that, as long as you assert that your app doesn't derive physical location from Wi-Fi device information, you don't need to declare the ACCESS_FINE_LOCATION permission anymore:

Figure 1. Flow chart to determine whether an app that targets Android 13 (API level 33) or higher can access Wi-Fi information.

Figure 2 shows the Wi-Fi access workflow on devices that run 12L or lower. Note the reliance on the ACCESS_FINE_LOCATION permission.

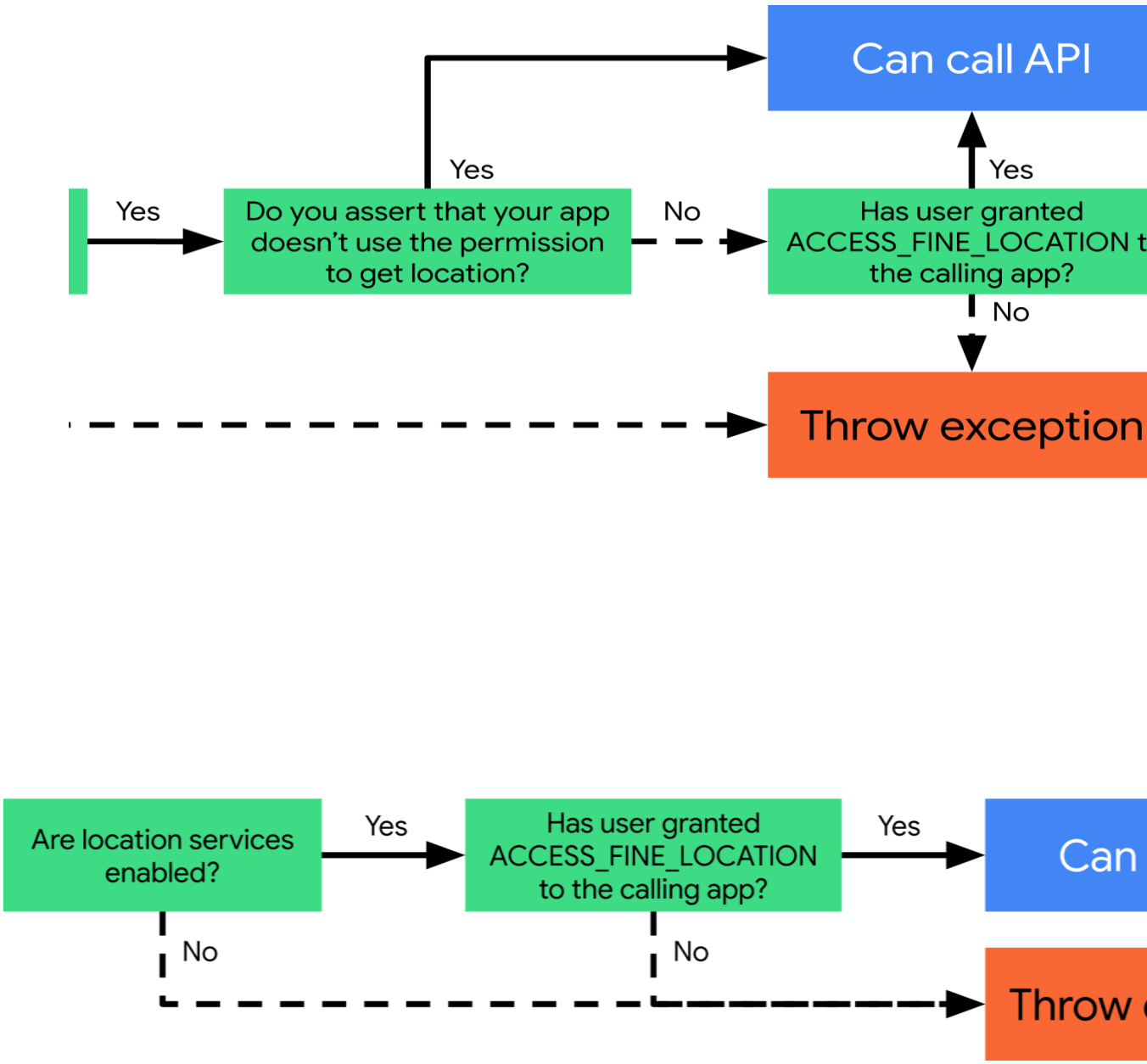

Figure 2. Flow chart to determine whether an app that targets 12L (API level 32) or lower can access Wi-Fi information.

Request permission to access nearby Wi-Fi devices

bookmark_border

Apps that target Android 13 (API level 33) or higher and manage Wi-Fi connections must request the NEARBY_WIFI_DEVICES runtime permission. This permission makes it easier to justify an app's access of nearby Wi-Fi devices; on previous versions of Android, these apps needed to declare the ACCESS_FINE_LOCATION permission instead.

Caution: If your app tries to call a Wi-Fi API without the proper permission, a **SecurityException** occurs.

Permission is part of the nearby devices group

The NEARBY_WIFI_DEVICES permission is part of the **Nearby devices** permission group. This group, added in Android 12 (API level 31), also includes permissions related to Bluetooth and Ultra-wideband. When you request any combination of permissions from this permission group, the system shows a single runtime dialog and asks the user to approve your app's access to nearby devices. In system settings, the user must enable and disable the **Nearby devices** permissions as a group; for example, users can't disable Wi-Fi access but keep Bluetooth access enabled for a given app.

Strongly assert that your app doesn't derive physical location

When you target Android 13 or higher, consider whether your app ever derives location information from Wi-Fi APIs; if not, you should strongly assert that. To make this assertion, set the usesPermissionFlags attribute to neverForLocation in your app's manifest file, as shown in the following code snippet. This process is similar to the one you do when you assert that Bluetooth device information is never used for location:

```
<manifest ...>
                                        <uses-permission
android:name="android.permission.NEARBY_WIFI_DEVICES"
            android:usesPermissionFlags="neverForLocation" />
  <application ...>
    ...
```

```
  </application>
</manifest>
```

Previous versions and some APIs require location permission

Several Wi-Fi APIs require the ACCESS_FINE_LOCATION permission, even when your app targets Android 13 or higher. Examples include the following methods from the WifiManager class:

- getScanResults()
- startScan()

Also, because the NEARBY_WIFI_DEVICES permission is available only on Android 13 and higher, you should keep any declarations for ACCESS_FINE_LOCATION to provide backward compatibility in your app. However, as long as your app doesn't otherwise rely on precise location information, you can set the maximum SDK version of this permission to 32, as shown in the following code snippet:

```
<manifest ...>
                                                    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"
          android:maxSdkVersion="32" />
  <application ...>
    ...
  </application>
</manifest>
```

Check for APIs that require the permission

If your app targets Android 13 or higher, you must declare the NEARBY_WIFI_DEVICES permission to call any of the following Wi-Fi APIs:

- WifiManager
- startLocalOnlyHotspot()

- WifiAwareManager
- attach()
- WifiAwareSession
- publish()
- subscribe()
- WifiP2pManager
- addLocalService()
- connect()
- createGroup()
- discoverPeers()
- discoverServices()
- requestDeviceInfo()
- requestGroupInfo()
- requestPeers()
- WifiRttManager
- startRanging()

Wi-Fi access workflows

Figure 1 shows the Wi-Fi access workflow on devices that run Android 13 or higher, for apps that target Android 13 or higher. Note that, as long as you assert that your app doesn't derive physical location from Wi-Fi device information, you don't need to declare the ACCESS_FINE_LOCATION permission anymore:

Figure 1. Flow chart to determine whether an app that targets Android 13 (API level 33) or higher can access Wi-Fi information.

Figure 2 shows the Wi-Fi access workflow on devices that run 12L or lower. Note the reliance on the ACCESS_FINE_LOCATION permission.

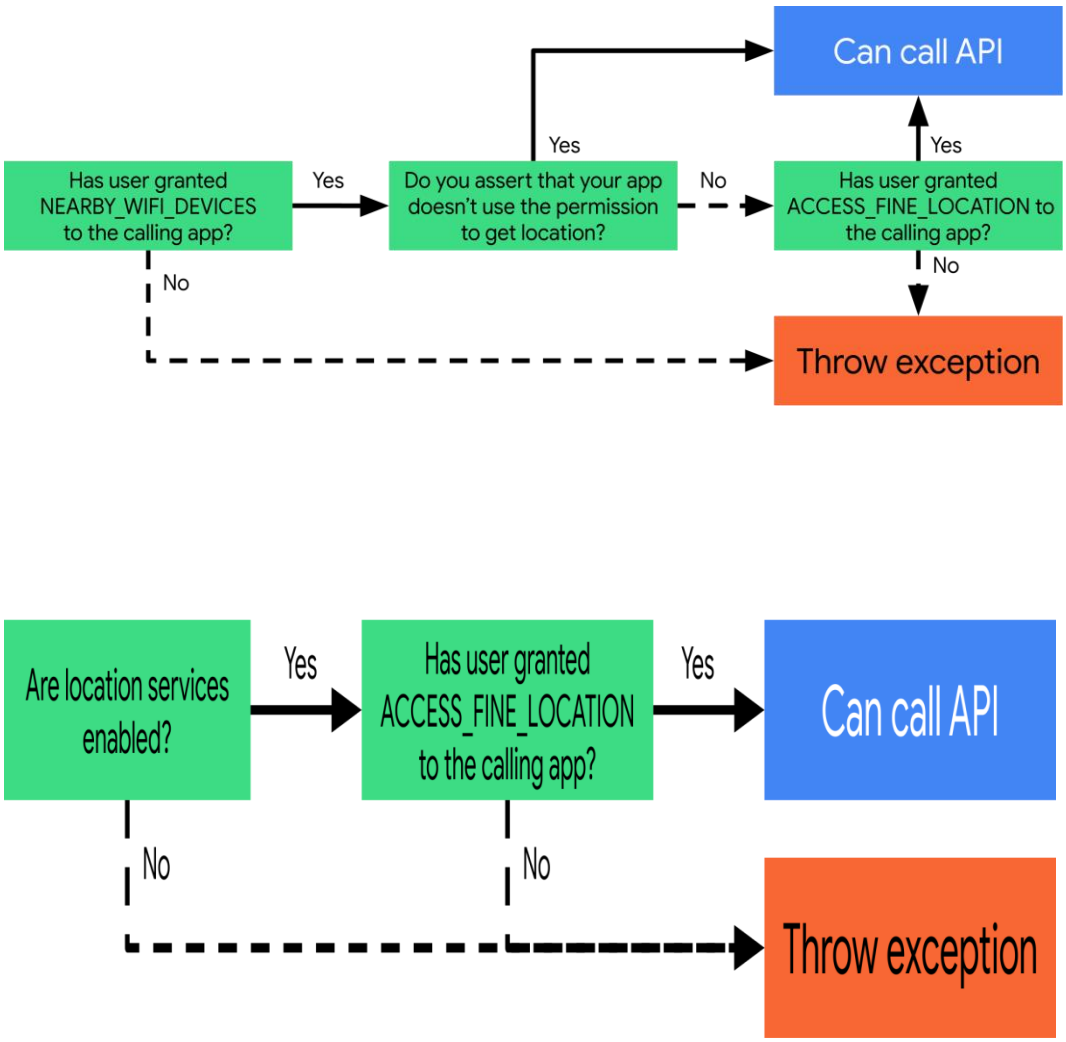

Figure 2. Flow chart to determine whether an app that targets 12L (API level 32) or lower can access Wi-Fi information.

Location Data

bookmark_border

One of the unique features of mobile applications is location awareness. Mobile users bring their devices with them everywhere, and adding location awareness to your app offers users a more contextual experience.

Code samples

The ApiDemos repository on GitHub includes samples that demonstrate the use of location on a map:

Java

- MyLocationDemoActivity: Using the My Location layer, including runtime permissions
- LocationSourceDemoActivity: Using a custom LocationSource
- CurrentPlaceDetailsOnMap: Finding the current location of an Android device and displaying details of the place (business or other point of interest) at that location. See the tutorial on showing current place details on a map.

Kotlin

- MyLocationDemoActivity: Using the My Location layer, including runtime permissions
- LocationSourceDemoActivity: Using a custom LocationSource
- CurrentPlaceDetailsOnMap: Finding the current location of an Android device and displaying details of the place (business or other point of interest) at that location. See the tutorial on showing current place details on a map.

Working with location data

The location data available to an Android device includes the current location of the device — pinpointed using a combination of technologies — the direction and method of movement, and whether the device has moved across a

predefined geographical boundary, or geofence. Depending upon the needs of your application, you can choose between several ways of working with location data:

- The **My Location** layer provides a simple way to display a device's location on the map. It does not provide data.
- The **Google Play services Location API** is recommended for all programmatic requests for location data.
- The LocationSource interface allows you to provide a custom location provider.

Location permissions

If your app needs to access the user's location, you must request permission by adding the relevant Android **location permissions** to your app.

Android offers two location permissions: ACCESS_COARSE_LOCATION and ACCESS_FINE_LOCATION. The permission you choose determines the accuracy of the location returned by the API.

- android.permission.ACCESS_COARSE_LOCATION – Allows the API to return the device's approximate location. The permission provides a device location estimate from location services, as described in the documentation about approximate location accuracy.
- android.permission.ACCESS_FINE_LOCATION – Allows the API to determine as precise a location as possible from the available location providers, including the Global Positioning System (GPS) as well as WiFi and mobile cell data.

Note: On Android 12 (API level 31) or higher, users can request that your app retrieve only approximate location information even when your app requests the **ACCESS_FINE_LOCATION** runtime permission. To handle this, both **ACCESS_FINE_LOCATION** and **ACCESS_COARSE_LOCATION** permissions

should be requested in a single runtime request. See User can grant only approxmate location.

Add the permissions to the app manifest

If approximate location is only needed for your app to function, then add the ACCESS_COARSE_LOCATION permission to your app's manifest file:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.myapp" >
  ...
                                                    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
  ...
</manifest>
```

However, if precise location is needed, then add both ACCESS_COARSE_LOCATION and ACCESS_FINE_LOCATION permissions to your app's manifest file:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.myapp" >
  ...
                                                    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
                                                    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
  ...
</manifest>
```

Request runtime permissions

Android 6.0 (Marshmallow) introduces a new model for handling permissions, which streamlines the process for users when they install and upgrade apps. If your app targets API level 23 or later, you can use the new permissions model.

If your app supports the new permissions model and the device is running Android 6.0 (Marshmallow) or later, the user does not have to grant any permissions when they install or upgrade the app. The app must check to see if it has the necessary permission at runtime, and request the permission if it does not have it. The system displays a dialog to the user asking for the permission.

For best user experience, it's important to request the permission in context. If location is essential to the functioning of your app, then you should request the location permission at app startup. A good way to do this is with a warm welcome screen or wizard that educates users about why the permission is required.

If the app requires the permission for only part of its functionality, then you should request the location permission at the time when the app performs the action that requires the permission.

The app must gracefully handle the case where the user does not grant permission. For example, if the permission is needed for a specific feature, the app can disable that feature. If the permission is essential for the app to function, the app can disable all its functionality and inform the user that they need to grant the permission.

The following code sample checks for permission using the AndroidX library before enabling the My Location layer. It then handles the result of the permission request by implementing the ActivityCompat.OnRequestPermissionsResultCallback from the Support library:

JavaKotlin

```
// Copyright 2020 Google LLC
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
```

```
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.


package com.example.mapdemo;

import android.Manifest.permission;
import android.annotation.SuppressLint;
import com.google.android.gms.maps.GoogleMap;
import
com.google.android.gms.maps.GoogleMap.OnMyLocationButtonClickListener;
import com.google.android.gms.maps.GoogleMap.OnMyLocationClickListener;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;

import android.Manifest;
import android.content.pm.PackageManager;
import android.location.Location;
import android.os.Bundle;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import android.widget.Toast;
```

```
/**
 * This demo shows how GMS Location can be used to check for changes to the users location.  The "My
 * Location" button uses GMS Location to set the blue dot representing the users location.
 * Permission for {@link android.Manifest.permission#ACCESS_FINE_LOCATION} and {@link
 * android.Manifest.permission#ACCESS_COARSE_LOCATION} are requested at run time. If either
 * permission is not granted, the Activity is finished with an error message.
 */
public class MyLocationDemoActivity extends AppCompatActivity
    implements
    OnMyLocationButtonClickListener,
    OnMyLocationClickListener,
    OnMapReadyCallback,
    ActivityCompat.OnRequestPermissionsResultCallback {

    /**
     * Request code for location permission request.
     *
     * @see #onRequestPermissionsResult(int, String[], int[])
     */
    private static final int LOCATION_PERMISSION_REQUEST_CODE = 1;

    /**
      * Flag indicating whether a requested permission has been denied after returning in {@link
     * #onRequestPermissionsResult(int, String[], int[])}.
     */
    private boolean permissionDenied = false;
```

```
  private GoogleMap map;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.my_location_demo);

    SupportMapFragment mapFragment =
                                                    (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);
  }

  @Override
  public void onMapReady(@NonNull GoogleMap googleMap) {
    map = googleMap;
    map.setOnMyLocationButtonClickListener(this);
    map.setOnMyLocationClickListener(this);
    enableMyLocation();
  }

  /**
   * Enables the My Location layer if the fine location permission has been
granted.
   */
  @SuppressLint("MissingPermission")
  private void enableMyLocation() {
    // 1. Check if permissions are granted, if so, enable the my location layer
                                if      (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
      == PackageManager.PERMISSION_GRANTED
                                    ||    ContextCompat.checkSelfPermission(this,
permission.ACCESS_COARSE_LOCATION)
```

```
        == PackageManager.PERMISSION_GRANTED) {
        map.setMyLocationEnabled(true);
        return;
      }

      // 2. Otherwise, request location permissions from the user.
                                    PermissionUtils.requestLocationPermissions(this,
LOCATION_PERMISSION_REQUEST_CODE, true);
   }

   @Override
   public boolean onMyLocationButtonClick() {
                        Toast.makeText(this, "MyLocation button clicked",
Toast.LENGTH_SHORT).show();
      // Return false so that we don't consume the event and the default behavior
still occurs
      // (the camera animates to the user's current position).
      return false;
   }

   @Override
   public void onMyLocationClick(@NonNull Location location) {
                    Toast.makeText(this, "Current location:\n" + location,
Toast.LENGTH_LONG).show();
   }

   @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions,
      @NonNull int[] grantResults) {
      if (requestCode != LOCATION_PERMISSION_REQUEST_CODE) {
                      super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
```

```
            return;
        }

        if (PermissionUtils.isPermissionGranted(permissions, grantResults,
            Manifest.permission.ACCESS_FINE_LOCATION) || PermissionUtils
            .isPermissionGranted(permissions, grantResults,
                Manifest.permission.ACCESS_COARSE_LOCATION)) {
            // Enable the my location layer if the permission has been granted.
            enableMyLocation();
        } else {
            // Permission was denied. Display an error message
                // Display the missing permission error dialog when the fragments
resume.
            permissionDenied = true;
        }
    }

    @Override
    protected void onResumeFragments() {
        super.onResumeFragments();
        if (permissionDenied) {
            // Permission was not granted, display error dialog.
            showMissingPermissionError();
            permissionDenied = false;
        }
    }

    /**
     * Displays a dialog with error message explaining that the location permission
is missing.
     */
    private void showMissingPermissionError() {
        PermissionUtils.PermissionDeniedDialog
```

```
        .newInstance(true).show(getSupportFragmentManager(), "dialog");
    }

}
```

MyLocationDemoActivity.java

The My Location layer

You can use the My Location layer and the My Location button to show your user their current position on the map. Call mMap.setMyLocationEnabled() to enable the My Location layer on the map.

Note: Before enabling the My Location layer, you must ensure that you have the required runtime location permission.

The following sample shows a simple usage of the My Location layer:

JavaKotlin

```
// Copyright 2020 Google LLC
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
```

```
// limitations under the License.

package com.google.maps.example;

import android.annotation.SuppressLint;
import android.location.Location;
import android.os.Bundle;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;

class MyLocationLayerActivity extends AppCompatActivity
  implements GoogleMap.OnMyLocationButtonClickListener,
  GoogleMap.OnMyLocationClickListener,
  OnMapReadyCallback {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_my_location);

    SupportMapFragment mapFragment =
                                                    (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);
  }

  @SuppressLint("MissingPermission")
```

```
    @Override
    public void onMapReady(GoogleMap map) {
        // TODO: Before enabling the My Location layer, you must request
        // location permission from the user. This sample does not include
        // a request for location permission.
        map.setMyLocationEnabled(true);
        map.setOnMyLocationButtonClickListener(this);
        map.setOnMyLocationClickListener(this);
    }

    @Override
    public void onMyLocationClick(@NonNull Location location) {
        Toast.makeText(this, "Current location:\n" + location, Toast.LENGTH_LONG)
            .show();
    }

    @Override
    public boolean onMyLocationButtonClick() {
        Toast.makeText(this, "MyLocation button clicked", Toast.LENGTH_SHORT)
            .show();
        // Return false so that we don't consume the event and the default behavior
still occurs
        // (the camera animates to the user's current position).
        return false;
    }
}
```

MyLocationLayerActivity.java

When the My Location layer is enabled, the My Location button appears in the top right corner of the map. When a user clicks the button, the camera centers the map on the current location of the device, if it is known. The location is

indicated on the map by a small blue dot if the device is stationary, or as a chevron if the device is moving.

The following screenshot shows the My Location button at top right and the My Location blue dot in the center of the map:

You can prevent the My Location button from appearing by calling UiSettings.setMyLocationButtonEnabled(false).

Your app can respond to the following events:

- If the user clicks the My Location button, your app receives an onMyLocationButtonClick() callback from the GoogleMap.OnMyLocationButtonClickListener.
- If the user clicks the My Location blue dot, your app receives an onMyLocationClick() callback from the GoogleMap.OnMyLocationClickListener.

The Google Play services Location API

The Google Play services Location API is the preferred method for adding location awareness to your Android application. It includes functionality that lets you:

- Determine the device location.
- Listen for location changes.
- Determine the mode of transportation, if the device is moving.
- Create and monitor predefined geographical regions, known as geofences.

The location APIs make it easy for you to build power efficient, location-aware applications. Like the Maps SDK for Android, the Location API is distributed as part of the Google Play services SDK. For more information on the Location API,

please refer to the Android training class Making Your App Location Aware or the Location API Reference. Code examples are included as part of the Google Play services SDK.