

ATLS 4120: Mobile Application Development

Week 15: Google Location

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>
You can get the last location using `getLastLocation()` or request regular updates using `requestLocationUpdates()`.

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.

Every Android app runs in a limited-access sandbox. If an app needs to use resources or information outside of its own sandbox, the app has to request the appropriate permission. You declare that your app needs a permission by listing the permission in the app manifest using a `<uses-permission>` element. You then must request that the user approve each permission 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.

- `checkSelfPermission()` checks if the user has granted permission
 - `PERMISSION_DENIED`
 - `PERMISSION_GRANTED`
- `requestPermissions()` requests the appropriate permission for your app
- `shouldShowRequestPermissionRationale()` returns true if the user has previously denied the request, and returns false if a user has denied a permission and selected the "Don't ask again" option in the permission request dialog, or if a device policy prohibits the permission.

Also, users are able to revoke the granted permission anytime through a device's settings so you always need to check to see if they've granted permission and request again if needed.

When the user responds to your app's permission request, the system invokes your app's `onRequestPermissionsResult()` method, passing it the user response. Your app has to override that method to find out whether the permission was granted. The callback is passed the same request code you passed to `requestPermissions()`.

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 and sync. To run on your device you might need the same version you used for maps if you changed that to match the version of Google Play Services.

implementation **'com.google.android.gms:play-services-location:17.0.0'**

Now let's update `onMapReady()` so we don't have the hard coded marker for Sydney.

```

@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;

    //set map type
    mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
}

```

Location Request

<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

FusedLocationProviderClient (Google Play services version 11.6.0 or higher)

The Fused Location Provider Client is one of the location APIs in Google Play services. It manages the underlying location technology such as analyzing 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. It also optimizes the device's use of battery power.

```

protected void startLocationUpdates() {
    // A request object to store parameters for requests to the
    FusedLocationProviderApi
    LocationRequest mLocationRequest = new LocationRequest();
    mLocationRequest.setInterval(1000); //set the desired interval
    for active location updates, in milliseconds
    mLocationRequest.setFastestInterval(500); //set the fastest
    interval for location updates, in milliseconds

    mLocationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_
    ACCURACY); //set priority of the request

    // Create LocationSettingsRequest object using location request
    LocationSettingsRequest.Builder builder = new
    LocationSettingsRequest.Builder();
    builder.addLocationRequest(mLocationRequest);
    LocationSettingsRequest locationSettingsRequest =
    builder.build();

    // Check whether the device's system settings are properly
    configured for the app's location needs.
    SettingsClient settingsClient =
    LocationServices.getSettingsClient(this);
    settingsClient.checkLocationSettings(locationSettingsRequest);

    // new Google API SDK v11 uses getFusedLocationProviderClient
    FusedLocationProviderClient mFusedLocationClient =
    LocationServices.getFusedLocationProviderClient(this);
    if (ContextCompat.checkSelfPermission(this,
    android.Manifest.permission.ACCESS_FINE_LOCATION) ==
    PackageManager.PERMISSION_GRANTED) {

```

```

        //request location updates
        mFusedLocationClient.requestLocationUpdates(mLocationRequest,
new LocationCallback() {
            @Override
            public void onLocationResult(LocationResult
locationResult) {
onLocationChanged(locationResult.getLastLocation());
            }
        },
        Looper.myLooper());
    }
}

```

Call `startLocationUpdates()` from `onCreate()` for pre API 23. API 23 and later will call it once permissions have been granted (later on). You can also do this based on the version.

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

Location Changes

We've set up our app to get regular location intervals but now we have to implement `onLocationChanged()` which is called whenever the user location changes and `onLocationResult` gets called.

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

```

// marker for current location
private Marker mCurrentLocationMarker;

public void onLocationChanged(Location 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(15));
}

```

You can also use custom pins or use the Geocoder class to show the location's address.

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

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

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

Zoom level 0 corresponds to the fully zoomed-out world view. Most areas support zoom levels up to 20, while more remote areas only support zoom levels up to 13. A zoom level of 11 is a nice in-between value that shows enough detail without getting crazy-close.

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 two strings to strings.xml that we'll use in our permission explanation.

```

<string name="dialogTitle">Location access</string>
<string name="dialogMessage">Location access needed for this app to
show your current location</string>

```

In MapsActivity.java 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) {
        //permission not 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.

            new AlertDialog.Builder(this)
                .setTitle(R.string.dialogTitle)

```

```

        .setMessage(R.string.dialogMessage)
        .setPositiveButton("OK", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface
dialogInterface, int i) {
                //Prompt the user once explanation has
                been shown

                ActivityCompat.requestPermissions(MapsActivity.this, new
                String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
                MY_PERMISSIONS_REQUEST_LOCATION);
            }
        })
        .create()
        .show();
    } else {
        //no explanation needed, request permissions
        ActivityCompat.requestPermissions(this, new
        String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
        MY_PERMISSIONS_REQUEST_LOCATION);
    }
}
//else permission has already been granted
else {
    mMap.setMyLocationEnabled(true);
    startLocationUpdates();
}
}

```

And call it in onMapReady()
`checkLocationPermission();`

`requestPermissions()` will then call `onRequestPermissionsResult()` and passes it the user's response and your request code.

```

@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) {

```

```

        mMap.setMyLocationEnabled(true);
        startLocationUpdates();
        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
}
}

```

To test all the conditions first deny access, you should see the permissions denied Toast. Then run the app again and you should get the AlertDialog(because access was previously denied). Then allow access and you'll see the permissions granted toast. Once you've granted permissions you won't be prompted again unless you go into Settings | Apps | Your App | Permissions and turn off location permission (or uninstall and reinstall the app on the device).

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