Location Services

Alexander Nelson
September 30, 2017

University of Arkansas - Department of Computer Science and Computer Engineering
Reminders
Project 2 is out!
Due Friday October 6th

Final Project Report 2 due Friday
Create a (temporary) App Name
Make your title be “AppName: Longer Description of Application”
Why is location important?
What about Mobile Computing?

1.2 Added Dimensions of Mobile Computing

- Multimodal and Variant UIs
- Large Variety of Platforms
- Active Behavior
- Limited Device Capabilities
- Limited Power Supply
- Location Awareness
- Wireless Connectivity

**Figure 1.3. Dimensions of Mobility.**

Mobility and Location

Advantage:
Accommodate logic based on locale
*Not unique to mobile*

Ex. POS systems and sales tax
Mobility and Location

**Advantage:**
Accomodate logic based on locale

*Not unique to mobile*

Instead:
Accomodate logic based on **changing** locale
How do mobile devices localize?

**GPS:**
Localization by message time-of-flight

**Cellular:**
Localization by time-of-flight or signal strength

**WiFi:**
Crowdsourced information from Wi-Fi base stations
Measure by ToF, RSSI, or Angle (for MIMO based routers)

\(^a\)Plus special and general relativity
Android and Localization
Two frameworks:

- android.location
- Google Play services location APIs (preferred)
Because the Google Play is preferred, we will use it.
To make sure it is installed, do the following

1. Start Android Studio.
2. On the Tools menu, click Android > SDK Manager.
3. Update the Android Studio SDK Manager: click SDK Tools, expand Support Repository, select Google Repository, and then click OK.
The Location Object is a Java Object that contains:

- Longitude
- Latitude
- Time
- Other fields
Android provides two location permission levels:

- Fine Location
- Coarse Location – Approximate, accurate to within a block

Request permission within manifest with:
<uses-permission android:name="android.permission.ACCESS_[FINE/COARSE]_LOCATION"/>
You will need access to the Google API for the FusedLocationProviderClient

Create a private object in your Class
e.g. private FusedLocationProviderClient mLocationClient

In onCreate() for the Component, get an instance:
mLocationClient = LocationServices.getFusedLocationProviderClient(this);
Getting Last Known Location
Getting the Last Known Location

Often, it is sufficient to get the last known location

```java
mFusedLocationClient.getLastLocation()
    .addOnSuccessListener(this, new OnSuccessListener<Location>() {
        @Override
        public void onSuccess(Location location) {
            // Got last known location. In some rare situations this can be null.
            if (location != null) {
                // Logic to handle location object
            }
        }
    });
```

Returns a (GoogleAPI)Task (an asynchronous operation, which represents a location)
Can attach an onSuccess() listener to get the Location object
It is possible for the Location object to be null
Situations where that is possible:

- Location is turned off in the device settings
  - Can be null even if the last location was previously retrieved because disabling location clears the cache
- Device has never recorded its location
- Google Play services on the device has restarted
  - Need to get access to the new instance of the Fused Location Provider
Location Settings
Your application may need to request changes to the Location settings.

Use the Settings Client to check which settings are enabled.
LocationRequest Object

LocationRequests object is used to store parameters for requests to the Location provider

Example Parameters

- `setInterval()` – Set rate in milliseconds at which you prefer to receive location updates (actual rate may be different)
- `setFastestInterval()` – Set the fastest rate at which your app can handle location updates
- `setPriority()` – Sets the priority of the requests
  - `PRIORITY_BALANCED_POWER_ACCURACY`
  - `PRIORITY_HIGH_ACCURACY`
  - `PRIORITY_LOW_POWER`
  - `PRIORITY_NO_POWER`
Example LocationRequest Object

```java
protected void createLocationRequest() {
    LocationRequest mLocationRequest = new LocationRequest();
    mLocationRequest.setInterval(10000);
    mLocationRequest.setFastestInterval(5000);
    mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
}
```
LocationSettingsRequest Object allows the application to check the location settings

Create an instance of a LocationSettingsRequest.Builder object
Then call .addLocationRequest() with a parameter of the LocateonRequest object created in the example
You can check if the Location setting is satisfied using the following code:

```java
LocationSettingsRequest.Builder builder = new LocationSettingsRequest.Builder();

// ...

SettingsClient client = LocationServices.getSettingsClient(this);
Task<LocationSettingsResponse> task = client.checkLocationSettings(builder.build());
```

When the (Google API)Task completes, the LocationSettingsResponse object can be queried with getLocationSettingsStates(), returning an object that has fields describing the LocationServices states.
Prompting a Change

If the settings do not match what your application needs
You will need to ask the user to change settings

Can be determined by querying the Status Codes during a failure
If the Task completes successful, it will enter this method:

task.addOnSuccessListener(this, new OnSuccessListener<LocationSettingsResponse>() {
    @Override
    public void onSuccess(LocationSettingsResponse locationSettingsResponse) {
        // All location settings are satisfied. The client can initialize
        // location requests here.
        // ...
    }
});
If the Task completes unsuccessFully, it will enter this method:

task.addOnFailureListener(this, new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        int statusCode = ((ApiException) e).getStatusCode();
        switch (statusCode) {
            case CommonStatusCodes.RESOLUTION_REQUIRED:
                // Location settings are not satisfied, but this can be fixed
                // by showing the user a dialog.
                try {
                    // Show the dialog by calling startResolutionForResult(),
                    // and check the result in onActivityResult().
                    ResolvableApiException resolvable = (ResolvableApiException) e;
                    resolvable.startResolutionForResult(
                        MainActivity.this,
                        REQUEST_CHECK_SETTINGS);
                } catch (IntentSender.SendIntentException sendEx) {
                    // Ignore the error.
                }
                break;
            case LocationSettingsStatusCodes.SETTINGS_CHANGE_UNAVAILABLE:
                // Location settings are not satisfied. However, we have no way
                // to fix the settings so we won't show the dialog.
                break;
        }
    }
});
The previous example shows creation of a dialog, which asks the users to request a change.

If the settings can’t be rectified, it instead hits the SETTINGS_CHANGE_UNAVAILABLE case, and does not prompt the user.
Requesting Location Updates
Some applications require updating the location (e.g. Maps). This can be done by polling the `getLastLocation()` method. Better method is to request the System to provide updates to the application.
Before you can receive updates, you should:

- Start with the last known location
  - It may take a bit before an update is posted, and your app can start with a coarse position
- Create a LocationRequest object
  - This way the system knows when to send updates
Create the Request

Should be started in `onResume()` and paused when not needed

```java
@Override
protected void onResume() {
    super.onResume();
    if (mRequestingLocationUpdates) {
        startLocationUpdates();
    }
}

private void startLocationUpdates() {
    mFusedLocationClient.requestLocationUpdates(mLocationRequest, mLocationCallback, null /* Looper */);
}
```

Location Callback object is passed, and provides an API for updates
Location Callback

The LocationCallback object passed to requestLocationUpdates() will be invoked through its onLocationResult() method.

```java
private LocationCallback mLocationCallback;

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...

    mLocationCallback = new LocationCallback() {
        @Override
        public void onLocationResult(LocationResult locationResult) {
            for (Location location : locationResult.getLocations()) {
                // Update UI with location data
                // ...
            }
        }
    };
}
```
Stop Location Updates

When you no longer need updates, you should unregister your location updates.

E.g. When activity goes out of focus, or task goes into background.

```java
@Override
protected void onPause() {
    super.onPause();
    stopLocationUpdates();
}

private void stopLocationUpdates() {
    mFusedLocationClient.removeLocationUpdates(mLocationCallback);
}
```
If you stop updates during `onStop()` instead of `onPause()` you will need to check during `onResume()` if location Services are running. Use a private Boolean in the class to track so you don’t override instances.
Geocoding
Geocoding/Reverse Geocoding

Often, a Long/Lat is not the best way to describe a location. Your user is more likely to know addresses.

Geocoding – Going from address → geographic location
Reverse Geocoding – Geographic location → address
To perform reverse geocoding

1. Get a geographic location
   - getLastLocation()
2. Define an IntentService for handling requests in manifest
3. Create Geocoder Object in the Service
4. Call geocoder.getFromLocation() within the Service
   - Returns a list of addresses
IntentService is a class which allows running a task on a background thread
Create a Service which extends IntentService
Location and ResultReceiver passed to the IntentService on creation using Intent.putExtras()
Push results from Geocoder to the ResultReceiver
The Geocoder object provides Reverse Geocoding services. Create an instance in the `onHandleIntent()` function in the extended `IntentService` class.

Geocoder constructor takes a context and a Locale:

- Can use default locale for the device.
Get the address

geocoder.getFromLocation()

- **Parameters:**
  - Latitude
  - Longitude
  - maxResults – Number of strings which describe the immediate area

- **Returns:**
  - List<Address> – maxResults length list of Address objects

There is a lot of error handling needed for this method
ResultReceiver – Class that allows event-based message passing
Send success/failure message from IntentService to requesting Activity
Example:

```java
public class FetchAddressIntentService extends IntentService {
    protected ResultReceiver mReceiver;
    // ... 
    private void deliverResultToReceiver(int resultCode, String message) {
        Bundle bundle = new Bundle();
        bundle.putString(Contacts.RESULT_DATA_KEY, message);
        mReceiver.send(resultCode, bundle);
    }
}
```
Receive Geocoding Results

In requesting Activity, use `onReceiveResult` inside ResultReceiver class.

Example:

```java
private void fetchAddressButtonHandler(View view) {
    mFusedLocationClient.getLastLocation()
        .addOnSuccessListener(this, new OnSuccessListener<Location>() {
            @Override
            public void onSuccess(Location location) {
                mLastKnownLocation = location;

                // In some rare cases the location returned can be null
                if (mLastKnownLocation == null) {
                    return;
                }

                if (!Geocoder.isPresent()) {
                    Toast.makeText(MainActivity.this,
                        R.string.no_geocoder_available,
                        Toast.LENGTH_LONG).show();
                    return;
                }

                // Start service and update UI to reflect new location
                startIntentService();
                updateUI();
            }
        });
}
```
Example Code (click to follow):

googlesamples GitHub
Geofences
Geofencing

Geofence – Combine user’s location with areas of interest
A geofence is composed of:

- A longitude
- A latitude
- A radius

These three values create a circular “fence” around a particular point of interest.
There may be up to 100 active geofences per device user.
Geofence Events

Geofences allow triggering of events by the Location Service.

Three event types:

- Enter event – when the user arrives within the fence
- Exit event – when the user leaves the fence
- Dwell event – when the user has been in the fence for a specified amount of time
Geofencing
Request Permissions:
Your app must request ACCESS_FINE_LOCATION in manifest

Create an IntentService to listen for geofence events

Create an instance of the LocationServices.GeofencingClient
mGeofencingClient = LocationServices.getGeofencingClient(this);
Create/Add Initial Geofences

Use Geofence.Builder() to create a geofence
Set desired longitude, latitude, and radius
Additionally, set duration and any transitions on which to alert

```java
mGeofenceList.add(new Geofence.Builder()
    // Set the request ID of the geofence. This is a string to identify this
    // geofence.
    .setRequestId(entry.getKey())

    .setCircularRegion(
        entry.getValue().latitude,
        entry.getValue().longitude,
        Constants.GEOFENCE_RADIUS_IN METERS
    )
    .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
    Geofence.GEOFENCE_TRANSITION_EXIT)
    .build());
```
Specify Geofences and Triggers

Use GeofencingRequest and the GeofencingRequestBuilder to specify which geofences to monitor for events.

```java
private GeofencingRequest getGeofencingRequest() {
    GeofencingRequest.Builder builder = new GeofencingRequest.Builder();
    builder.setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER);
    builder.addGeofences(mGeofenceList);
    return builder.build();
}
```

INITIAL_TRIGGER_ENTER – Tells device to trigger the GEOFENCE_TRANSITION_ENTER if device is already inside radius when fence is created.
The LocationServices will trigger an Intent on Geofence events. The Intent **should not** create an Activity or Fragment (bad UX).

IntentService is a better way to handle the Intent to perform a task on these events.
public class MainActivity extends AppCompatActivity {

    // ...

    private PendingIntent getGeofencePendingIntent() {
        // Reuse the PendingIntent if we already have it.
        if (mGeofencePendingIntent != null) {
            return mGeofencePendingIntent;
        }

        Intent intent = new Intent(this, GeofenceTransitionsIntentService.class);
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when
        // calling addGeofences() and removeGeofences().
        mGeofencePendingIntent = PendingIntent.getService(this, 0, intent, PendingIntent.
        FLAG_UPDATE_CURRENT);
        return mGeofencePendingIntent;
    }
}
Add Geofences

To add geofences, use the `GeofencingClient.addGeofences()` method.

Provide the `GeofencingRequest` object, and the `PendingIntent`.

```java
mGeofencingClient.addGeofences(getGeofencingRequest(), getGeofencePendingIntent())
    .addOnSuccessListener(this, new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void aVoid) {
            // Geofences added
            // ...
        }
    })
    .addOnFailureListener(this, new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            // Failed to add geofences
            // ...
        }
    });
```
Handle Geofence Transitions

There are many things that an IntentService may do with Geofence Transitions.
A common scenario is to post a notification on a Geofence Event.

Handle Error:

```java
public class GeofenceTransitionsIntentService extends IntentService {
    // ...
    protected void onHandleIntent(Intent intent) {
        GeofencingEvent geofencingEvent = GeofencingEvent.fromIntent(intent);
        if (geofencingEvent.hasError()) {
            String errorMessage = GeofenceErrorMessages.getErrorString(this,
                geofencingEvent.getErrorCode);
            Log.e(TAG, errorMessage);
            return;
        }
    }
```
// Get the transition type.
int geofenceTransition = geofencingEvent.getGeofenceTransition();

// Test that the reported transition was of interest.
if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER ||
    geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {

    // Get the geofences that were triggered. A single event can trigger
    // multiple geofences.
    List<Geofence> triggeringGeofences = geofencingEvent.getTriggeringGeofences();

    // Get the transition details as a String.
    String geofenceTransitionDetails = getGeofenceTransitionDetails(
            this,
            geofenceTransition,
            triggeringGeofences
        );

    // Send notification and log the transition details.
    sendNotification(geofenceTransitionDetails);
    Log.i(TAG, geofenceTransitionDetails);
} else {
    // Log the error.
    Log.e(TAG,(getString(R.string.geofence_transition_invalid_type,
            geofenceTransition)));
}
You can stop Geofence Monitoring using the GeofencingClient. Can remove by requestId or the PendingIntent associated with the request:

```java
mGeofencingClient.removeGeofences(mGeofencePendingIntent)
```

The `removeGeofences` is a Task, and has `onSuccess` and `onFailure` methods which can be used to make sure geofences are removed.
Things to remember for Geofences:

- Reduce Power Consumption
- Choose optimal radius for geofence
- Use dwell transition type to reduce alert spam
- Re-register geofences only when required, e.g.
  - Device rebooted
  - App reinstalled
  - App data cleared
  - Google Play services data cleared