Receiving Location Updates

THIS LESSON TEACHES YOU TO

YOU SHOULD ALSO READ

- Setup Google Play Services SDK
- Retrieving the Current Location

TRY IT OUT

Download the sample

LocationUpdates.zip

If your app does navigation or tracking, you probably want to get the user's location at regular intervals. While you can do this with `LocationClient.getLastLocation()` `(/reference/com/google/android/gms/location/LocationClient.html#getLastLocation())`, a more direct approach is to request periodic updates from Location Services. In response, Location Services automatically updates your app with the best available location, based on the currently-available location providers such as WiFi and GPS.

To get periodic location updates from Location Services, you send a request using a location client. Depending on the form of the request, Location Services either invokes a callback method and passes in a `Location` `(/reference/android/location/Location.html)` object, or issues an `Intent` `(/reference/android/content/Intent.html)` that contains the location in its extended data. The accuracy and frequency of the updates are affected by the location permissions you've requested and the parameters you pass to Location Services with the request.

Specify App Permissions

Apps that use Location Services must request location permissions. Android has two location permissions, `ACCESS_COARSE_LOCATION` `(/reference/android/Manifest.permission.html#ACCESS_COARSE_LOCATION)` and `ACCESS_FINE_LOCATION` `(/reference/android/Manifest.permission.html#ACCESS_FINE_LOCATION)`. The permission you choose affects the accuracy of the location updates you receive. For example, If you request only coarse location permission, Location Services obfuscates the updated location to an accuracy that's roughly equivalent to a city block.

Requesting `ACCESS_FINE_LOCATION` `(/reference/android/Manifest.permission.html#ACCESS_FINE_LOCATION)` implies a request for `ACCESS_COARSE_LOCATION` `(/reference/android/Manifest.permission.html#ACCESS_COARSE_LOCATION)`.

For example, to add the coarse location permission to your manifest, insert the following as a child element of the `<manifest>` `(/guide/topics/manifest/manifest-element.html)` element:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Check for Google Play Services

Location Services is part of the Google Play services APK. Since it's hard to anticipate the state of the user's device, you should always check that the APK is installed before you attempt to connect to Location Services. To check that the APK is installed, call `GooglePlayServicesUtil.isGooglePlayServicesAvailable()` (/reference/com/google/android/gms/common/GooglePlayServicesUtil.html#isGooglePlayServicesAvailable(android.content.Context)), which returns one of the integer result codes listed in the API reference documentation. If you encounter an error, call `GooglePlayServicesUtil.getErrorDialog()` (/reference/com/google/android/gms/common/GooglePlayServicesUtil.html#getErrorDialog(int, android.app.Activity, int)) to retrieve localized dialog that prompts users to take the correct action, then display the dialog in a `DialogFragment` (/reference/android/support/v4/app/DialogFragment.html). The dialog may allow the user to correct the problem, in which case Google Play services may send a result back to your activity. To handle this result, override the method `onActivityResult()` (/reference/android/support/v4/app/FragmentActivity.html#onActivityResult(int, int, android.content.Intent))

> **Note:** To make your app compatible with platform version 1.6 and later, the activity that displays the `DialogFragment` (/reference/android/support/v4/app/DialogFragment.html) must subclass `FragmentActivity` (/reference/android/support/v4/app/FragmentActivity.html) instead of `Activity` (/reference/android/app/Activity.html). Using `FragmentActivity` (/reference/android/support/v4/app/FragmentActivity.html) also allows you to call `getSupportFragmentManager()` (/reference/android/support/v4/app/FragmentActivity.html#getSupportFragmentManager()) to display the `DialogFragment` (/reference/android/support/v4/app/DialogFragment.html).

Since you usually need to check for Google Play services in more than one place in your code, define a method that encapsulates the check, then call the method before each connection attempt. The following snippet contains all of the code required to check for Google Play services:

```java
public class MainActivity extends FragmentActivity {
    ...
    // Global constants
    /*
     * Define a request code to send to Google Play services
     * This code is returned in Activity.onActivityResult
     */
    private final static int
            CONNECTION_FAILURE_RESOLUTION_REQUEST = 9000;
    ...
    // Define a DialogFragment that displays the error dialog
    public static class ErrorDialogFragment extends DialogFragment {
        // Global field to contain the error dialog
        private Dialog mDialog;
        // Default constructor. Sets the dialog field to null
        public ErrorDialogFragment() {
            super();
            mDialog = null;
        }
        // Set the dialog to display
        public void setDialog(Dialog dialog) {
            mDialog = dialog;
        }
        // Return a Dialog to the DialogFragment.
        @Override
        public Dialog onCreateDialog(Bundle savedInstanceState) {
            return mDialog;
        }
    }
    ...
```

```
/*
 * Handle results returned to the FragmentActivity
 * by Google Play services
 */
@Override
protected void onActivityResult(
        int requestCode, int resultCode, Intent data) {
    // Decide what to do based on the original request code
    switch (requestCode) {
        ...
        case CONNECTION_FAILURE_RESOLUTION_REQUEST :
        /*
         * If the result code is Activity.RESULT_OK, try
         * to connect again
         */
            switch (resultCode) {
                case Activity.RESULT_OK :
                /*
                 * Try the request again
                 */
                ...
                break;
            }
        ...
    }
    ...
}
...
private boolean servicesConnected() {
    // Check that Google Play services is available
    int resultCode =
            GooglePlayServicesUtil.
                    isGooglePlayServicesAvailable(this);
    // If Google Play services is available
    if (ConnectionResult.SUCCESS == resultCode) {
        // In debug mode, log the status
        Log.d("Location Updates",
                "Google Play services is available.");
        // Continue
        return true;
    // Google Play services was not available for some reason
    } else {
        // Get the error code
        int errorCode = connectionResult.getErrorCode();
        // Get the error dialog from Google Play services
        Dialog errorDialog = GooglePlayServicesUtil.getErrorDialog(
                errorCode,
                this,
                CONNECTION_FAILURE_RESOLUTION_REQUEST);
        // If Google Play services can provide an error dialog
        if (errorDialog != null) {
            // Create a new DialogFragment for the error dialog
            ErrorDialogFragment errorFragment =
                    new ErrorDialogFragment();
            // Set the dialog in the DialogFragment
            errorFragment.setDialog(errorDialog);
            // Show the error dialog in the DialogFragment
            errorFragment.show(
                    getSupportFragmentManager(),
                    "Location Updates");
        }
```

```
            }
        }
        ...
    }
```

Snippets in the following sections call this method to verify that Google Play services is available.

Define Location Services Callbacks

Before you request location updates, you must first implement the interfaces that Location Services uses to communicate connection status to your app:

`ConnectionCallbacks`
: Specifies methods that Location Services calls when a location client is connected or disconnected.

`OnConnectionFailedListener`
: Specifies a method that Location Services calls if an error occurs while attempting to connect the location client. This method uses the previously-defined `showErrorDialog` method to display an error dialog that attempts to fix the problem using Google Play services.

The following snippet shows how to specify the interfaces and define the methods:

```
public class MainActivity extends FragmentActivity implements
        GooglePlayServicesClient.ConnectionCallbacks,
        GooglePlayServicesClient.OnConnectionFailedListener {
    ...
    /*
     * Called by Location Services when the request to connect the
     * client finishes successfully. At this point, you can
     * request the current location or start periodic updates
     */
    @Override
    public void onConnected(Bundle dataBundle) {
        // Display the connection status
        Toast.makeText(this, "Connected", Toast.LENGTH_SHORT).show();
    }
    ...
    /*
     * Called by Location Services if the connection to the
     * location client drops because of an error.
     */
    @Override
    public void onDisconnected() {
        // Display the connection status
        Toast.makeText(this, "Disconnected. Please re-connect.",
                Toast.LENGTH_SHORT).show();
    }
    ...
    /*
     * Called by Location Services if the attempt to
     * Location Services fails.
     */
    @Override
    public void onConnectionFailed(ConnectionResult connectionResult) {
        /*
         * Google Play services can resolve some errors it detects.
         * If the error has a resolution, try sending an Intent to
         * start a Google Play services activity that can resolve
         * error.
         */
        if (connectionResult.hasResolution()) {
            try {
                // Start an Activity that tries to resolve the error
                connectionResult.startResolutionForResult(
                        this,
                        CONNECTION_FAILURE_RESOLUTION_REQUEST);
                /*
                * Thrown if Google Play services canceled the original
                * PendingIntent
                */
            } catch (IntentSender.SendIntentException e) {
                // Log the error
                e.printStackTrace();
            }
        } else {
            /*
             * If no resolution is available, display a dialog to the
             * user with the error.
             */
            showErrorDialog(connectionResult.getErrorCode());
        }
    }
    ...
}
```

Define the location update callback

Location Services sends location updates to your app either as an Intent (/reference/android/content/Intent.html) or as an argument passed to a callback method you define. This lesson shows you how to get the update using a callback method, because that pattern works best for most use cases. If you want to receive updates in the form of an Intent (/reference/android/content/Intent.html), read the lesson Recognizing the User's Current Activity (activity-recognition.html), which presents a similar pattern.

The callback method that Location Services invokes to send a location update to your app is specified in the LocationListener (/reference/com/google/android/gms/location/LocationListener.html) interface, in the method onLocationChanged((/reference/com/google/android/gms/location/LocationL er.html#onLocationChanged(android.location.Location)). The incoming argument is a Location (/refe e/android/location/Location.html) object containing the location's latitude and longitude. The following snippet shows how to specify the interface and define the method:

```
public class MainActivity extends FragmentAct     y implements
        GooglePlayServicesClient.ConnectionC      cks,
        GooglePlayServicesClient.OnConnectionFailedListener,
        LocationLister
    ...
    // Define the cal                             location updates
    @Override
    public void onLoca                            on) {
        // Report to t                            updated
        String msg = "
                Double                            ude()) + "," +
                Double                            tude());
        Toast.makeText                            ORT).show();
    }
    ...
}
```

Now that you have the callbac uest for location updates. The first step is to specify the parameters that cc

Activity

Best Practices for
User Experience & UI

Best Practices for
User Input

Best Practices for
Performance

Best Practices for
Security & Privacy

Using Google Play to
Distribute & Monetize

Specify Update Parameters

Location Services allows you to control the interval betwee.. updates and the location accuracy you want, by setting the values in a LocationRequest (/reference/com/google/android/gms/location/LocationR t.html) object and then sending this object as part of your request to start updates.

First, set the following interval parameters:

Update interval
: Set by LocationRequest.setInterval(). This me sets the rate in milliseconds at which your app prefers to receive location updates. If no other apps are receiving updates from Location Services, your app will receive updates at this rate.

Fastest update interval
: Set by LocationRequest.setFastestInterval(). This method sets the **fastest** rate in milliseconds at which your app can handle location updates. You need to set this rate because other apps also affect the rate at which updates are sent. Location Services sends out updates at the fastest rate that any app requested by calling LocationRequest.setInterval(). If this rate is faster than your app can handle, you may encounter problems with UI flicker or data overflow. To prevent this, call LocationRequest.setFastestInterval() to set an upper limit to the update rate.

  Calling LocationRequest.setFastestInterval() (/reference/com/google/android/gms/location/LocationRequest.html#setFastestInterval(long)) also helps to save power. When you request a preferred update rate by calling LocationRequest.setInterval() (/reference/com/google/android/gms/location/LocationRequest.html#setInterval(long)), and a maximum rate by calling LocationRequest.setFastestInterval() (/reference/com/google/android/gms/location/LocationRequest.html#setFastestInterval(long)), then your app gets the same update rate as the fastest rate in the system. If other apps have requested a faster rate, you get the benefit of a faster rate. If no other apps have a faster rate request outstanding, your app receives updates at the rate you specified with LocationRequest.setInterval() (/reference/com/google/android/gms/location/LocationRequest.html#setInterval(long)).

Next, set the accuracy parameter. In a foreground app, you need constant location updates with high accuracy, so use the setting LocationRequest.PRIORITY_HIGH_ACCURACY (/reference/com/google/android/gms/location/LocationRequest.html#PRIORITY_HIGH_ACCURACY).

The following snippet shows how to set the update interval and accuracy in onCreate() (/reference/android/support/v4/app/FragmentActivity.html#onCreate(android.os.Bundle)):

```
public class MainActivity extends FragmentActivity implements
        GooglePlayServicesClient.ConnectionCallbacks,
        GooglePlayServicesClient.OnConnectionFailedListener,
        LocationListener {
    ...
    // Global constants
    ...
    // Milliseconds per second
    private static final int MILLISECONDS_PER_SECOND = 1000;
    // Update frequency in seconds
    public static final int UPDATE_INTERVAL_IN_SECONDS = 5;
    // Update frequency in milliseconds
    private static final long UPDATE_INTERVAL =
            MILLISECONDS_PER_SECOND * UPDATE_INTERVAL_IN SECONDS;
    // The fastest update frequency, in seconds
    private static final int FASTEST_INTERVAL_IN_SECONDS = 1;
    // A fast frequency ceiling in milliseconds
    private static final long FASTEST_INTERVAL =
            MILLISECONDS_PER_SECOND * FASTEST_INTERVAL_IN_SECONDS;
    ...
    // Define an object that holds accuracy and frequency parameters
    LocationResult mLocationRequest;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Create the LocationRequest object
        mLocationRequest = LocationRequest.create();
        // Use high accuracy
        mLocationRequest.setPriority(
                LocationRequest.PRIORITY_HIGH_ACCURACY);
        // Set the update interval to 5 seconds
        mLocationRequest.setInterval(UPDATE_INTERVAL);
        // Set the fastest update interval to 1 second
        mLocationRequest.setFastestInterval(FASTEST_INTERVAL);
        ...
    }
    ...
}
```

Note: If your app accesses the network or does other long-running work after receiving a location update, adjust the fastest interval to a slower value. This prevents your app from receiving updates it can't use. Once the long-running work is done, set the fastest interval back to a fast value.

Start Location Updates

To send the request for location updates, create a location client in onCreate() (/reference/android/support/v4/app/FragmentActivity.html#onCreate(android.os.Bundle)), then connect it and make the request by calling requestLocationUpdates() (/reference/com/google/android/gms/location/LocationClient.html#requestLocationUpdates(com.google.android.gms.location.LocationRequest, com.google.android.gms.location.LocationListener)). Since your client must be connected for your app to receive updates, you should connect the client and make the request in onStart() (/reference/android/support/v4/app/FragmentActivity.html#onStart()). This ensures that you always have a valid, connected client while your app is visible.

Remember that the user may want to turn off location updates for various reasons. You should provide a way for the user to do this, and you should ensure that you don't start updates in onStart() (/reference/android/support/v4/app/FragmentActivity.html#onStart()) if updates were previously turned off. To track the user's preference, store it in your app's SharedPreferences (/reference/android/content/SharedPreferences.html) in onPause() (/reference/android/support/v4/app/FragmentActivity.html#onPause()) and retrieve it in onResume() (/reference/android/support/v4/app/FragmentActivity.html#onResume()).

The following snippet shows how to set up the client in onCreate() (/reference/android/support/v4/app/FragmentActivity.html#onCreate(android.os.Bundle)), and how to connect it and request updates in onStart() (/reference/android/support/v4/app/FragmentActivity.html#onStart()):

```
public class MainActivity extends FragmentActivity implements
        GooglePlayServicesClient.ConnectionCallbacks,
        GooglePlayServicesClient.OnConnectionFailedListener,
        LocationListener {
    ...
    // Global variables
    ...
    LocationClient mLocationClient;
    boolean mUpdatesRequested;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // Open the shared preferences
        mPrefs = getSharedPreferences("SharedPreferences",
                Context.MODE_PRIVATE);
        // Get a SharedPreferences editor
        mEditor = mPrefs.edit();
        /*
         * Create a new location client, using the enclosing class to
         * handle callbacks.
         */
        mLocationClient = new LocationClient(this, this, this);
        // Start with updates turned off
        mUpdatesRequested = false;
        ...
    }
    ...
    @Override
    protected void onPause() {
        // Save the current setting for updates
        mEditor.putBoolean("KEY_UPDATES_ON", mUpdatesRequested);
        mEditor.commit();
        super.onPause();
    }
    ...
    @Override
    protected void onStart() {
        ...
        mLocationClient.connect();
    }
    ...
    @Override
    protected void onResume() {
        /*
         * Get any previous setting for location updates
         * Gets "false" if an error occurs
         */
        if (mPrefs.contains("KEY_UPDATES_ON")) {
            mUpdatesRequested =
                    mPrefs.getBoolean("KEY_UPDATES_ON", false);

        // Otherwise, turn off location updates
        } else {
            mEditor.putBoolean("KEY_UPDATES_ON", false);
            mEditor.commit();
        }
    }
    ...
}
```

For more information about saving preferences, read Saving Key-Value Sets (/training/basics/data-storage/shared-preferences.html).

Stop Location Updates

To stop location updates, save the state of the update flag in onPause() (/reference/android/support/v4/app/FragmentActivity.html#onPause()), and stop updates in onStop() (/reference/android/support/v4/app/FragmentActivity.html#onStop()) by calling removeLocationUpdates(LocationListener) (/reference/com/google/android/gms/location/LocationClient.html#removeLocationUpdates(com.google.android.gms.location.LocationListener)). For example:

```
public class MainActivity extends FragmentActivity implements
        GooglePlayServicesClient.ConnectionCallbacks,
        GooglePlayServicesClient.OnConnectionFailedListener,
        LocationListener {
    ...
    /*
     * Called when the Activity is no longer visible at all.
     * Stop updates and disconnect.
     */
    @Override
    protected void onStop() {
        // If the client is connected
        if (mLocationClient.isConnected()) {
            /*
             * Remove location updates for a listener.
             * The current Activity is the listener, so
             * the argument is "this".
             */
            removeLocationUpdates(this);
        }
        /*
         * After disconnect() is called, the client is
         * considered "dead".
         */
        mLocationClient.disconnect();
        super.onStop();
    }
    ...
}
```

You now have the basic structure of an app that requests and receives periodic location updates. You can combine the features described in this lesson with the geofencing, activity recognition, or reverse geocoding features described in other lessons in this class.

The next lesson, Displaying a Location Address (display-address.html), shows you how to use the current location to display the current street address.