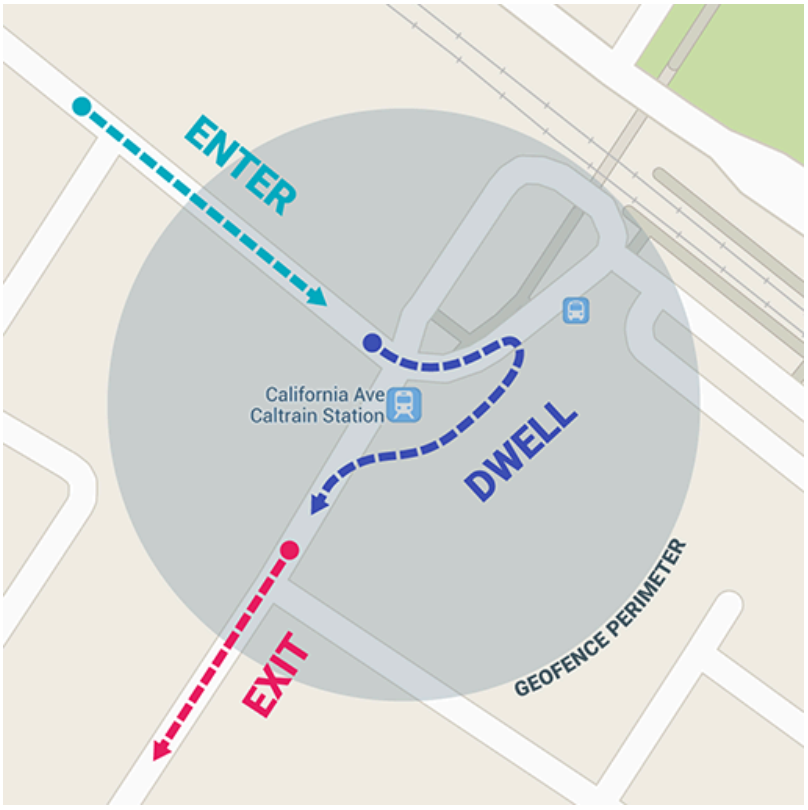
Create and monitor geofences

Archived: 2026-04-05 22:40:56 UTC

Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence, creating a circular area, or fence, around the location of interest.

You can have multiple active geofences, with a limit of 100 per app, per device user. For each geofence, you can ask Location Services to send you entrance and exit events, or you can specify a duration within the geofence area to wait, or *dwell*, before triggering an event. You can limit the duration of any geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.



This lesson shows you how to add and remove geofences, and then listen for geofence transitions using a `BroadcastReceiver`.

Note: On Wear devices, the Geofencing APIs don't make efficient use of power. We don't recommend these APIs on Wear. Read Conserve power and battery for more information.

Set up for geofence monitoring

The first step in requesting geofence monitoring is to request the necessary permissions. To use geofencing, your app must request the following:

- `ACCESS_FINE_LOCATION`
- `ACCESS_BACKGROUND_LOCATION` if your app targets Android 10 (API level 29) or higher

To learn more, see the guide on how to request location permissions.

If you want to use a `BroadcastReceiver` to listen for geofence transitions, add an element specifying the service name. This element must be a child of the `<application>` element:

```
<application
   android:allowBackup="true">
   ...
   <receiver android:name=".GeofenceBroadcastReceiver"/>
<application/>
```

To access the location APIs, you need to create an instance of the Geofencing client. To learn how to connect your client:

```
lateinit var geofencingClient: GeofencingClient

override fun onCreate(savedInstanceState: Bundle?) {
    // ...
    geofencingClient = LocationServices.getGeofencingClient(this)
}
```

```
private GeofencingClient geofencingClient;

@Override
public void onCreate(Bundle savedInstanceState) {
    // ...
    geofencingClient = LocationServices.getGeofencingClient(this);
}
```

Your app needs to create and add geofences using the location API's builder class for creating Geofence objects, and the convenience class for adding them. Also, to handle the intents sent from Location Services when geofence transitions occur, you can define a `PendingIntent` as shown in this section.

Note: On single-user devices, there is a limit of 100 geofences per app. For multi-user devices, the limit is 100 geofences per app per device user.

Create geofence objects

First, use `Geofence.Builder` to create a geofence, setting the desired radius, duration, and transition types for the geofence. For example, to populate a list object:

```
geofenceList.add(Geofence.Builder()
    // Set the request ID of the geofence. This is a string to identify this
    // geofence.
    .setRequestId(entry.key)

    // Set the circular region of this geofence.
    .setCircularRegion(
        entry.value.latitude,
        entry.value.longitude,
        Constants.GEOFENCE_RADIUS_IN_METERS
    )

    // Set the expiration duration of the geofence. This geofence gets automatically
    // removed after this period of time.
    .setExpirationDuration(Constants.GEOFENCE_EXPIRATION_IN_MILLISECONDS)

    // Set the transition types of interest. Alerts are only generated for these
    // transition. We track entry and exit transitions in this sample.
    .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER or Geofence.GEOFENCE_TRANSITION_EXIT)

    // Create the geofence.
    .build())
```

```
geofenceList.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());
```

This example pulls data from a constants file. In actual practice, apps might dynamically create geofences based on the user's location.

Specify geofences and initial triggers

The following snippet uses the `GeofencingRequest` class and its nested `GeofencingRequestBuilder` class to specify the geofences to monitor and to set how related geofence events are triggered:

```
private fun getGeofencingRequest(): GeofencingRequest {
    return GeofencingRequest.Builder().apply {
        setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER)
        addGeofences(geofenceList)
    }.build()
}
```

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

This example shows the use of two geofence triggers. The `GEOFENCE_TRANSITION_ENTER` transition triggers when a device enters a geofence, and the `GEOFENCE_TRANSITION_EXIT` transition triggers when a device exits a geofence. Specifying `INITIAL_TRIGGER_ENTER` tells Location services that `GEOFENCE_TRANSITION_ENTER` should be triggered if the device is already inside the geofence.

In many cases, it may be preferable to use instead `INITIAL_TRIGGER_DWELL`, which triggers events only when the user stops for a defined duration within a geofence. This approach can help reduce "alert spam" resulting from large numbers notifications when a device briefly enters and exits geofences. Another strategy for getting best results from your geofences is to set a minimum radius of 100 meters. This helps account for the location accuracy of typical Wi-Fi networks, and also helps reduce device power consumption.

Define a broadcast receiver for geofence transitions

An `Intent` sent from Location Services can trigger various actions in your app, but you should *not* have it start an activity or fragment, because components should only become visible in response to a user action. In many cases, a `BroadcastReceiver` is a good way to handle a geofence transition. A `BroadcastReceiver` gets updates when an event occurs, such as a transition into or out of a geofence, and can start long-running background work.

The following snippet shows how to define a `PendingIntent` that starts a `BroadcastReceiver`:

```
class MainActivity : AppCompatActivity() {

    // ...

    private val geofencePendingIntent: PendingIntent by lazy {
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when calling
        // addGeofences() and removeGeofences().
```

```
        val flags = PendingIntent.FLAG_UPDATE_CURRENT
        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.S) {
            // Starting on Android S+ the pending intent has to be mutable.
            flags or PendingIntent.FLAG_MUTABLE
        }
        val intent = Intent(this, GeofenceBroadcastReceiver::class.java)
        PendingIntent.getBroadcast(this, 0, intent, flags)
    }

}
```

```
public class MainActivity extends AppCompatActivity {

    // ...

    private PendingIntent getGeofencePendingIntent() {
        // Reuse the PendingIntent if we already have it.
        if (geofencePendingIntent != null) {
          return geofencePendingIntent;
        }
        // We use FLAG_UPDATE_CURRENT so that we get the same pending intent back when calling
        // addGeofences() and removeGeofences().
        int flags = PendingIntent.FLAG_UPDATE_CURRENT;
        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.S) {
            // Starting on Android S+ the pending intent has to be mutable.
            flags |= PendingIntent.FLAG_MUTABLE;
        }
        Intent intent = new Intent(this, GeofenceBroadcastReceiver.class);
        geofencePendingIntent = PendingIntent.getBroadcast(this, 0, intent, flags);
        return geofencePendingIntent;
   }

}
```

Add geofences

To add geofences, use the `GeofencingClient.addGeofences()` method. Provide the `GeofencingRequest` object, and the `PendingIntent`. The following snippet demonstrates processing the results:

```
geofencingClient?.addGeofences(getGeofencingRequest(), geofencePendingIntent)?.run {
    addOnSuccessListener {
        // Geofences added
        // ...
    }
    addOnFailureListener {
```

```
        // Failed to add geofences
        // ...
    }
}
```

```
geofencingClient.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

When Location Services detects that the user has entered or exited a geofence, it sends out the `Intent` contained in the `PendingIntent` you included in the request to add geofences. A broadcast receiver like `GeofenceBroadcastReceiver` notices that the `Intent` was invoked and can then obtain the geofencing event from the intent, determine the type of Geofence transition(s), and determine which of the defined geofences was triggered. The broadcast receiver can direct an app to start performing background work or, if desired, send a notification as output.

Note: On Android 8.0 (API level 26) and higher, if an app is running in the background while monitoring a geofence, then the device responds to geofencing events every couple of minutes. To learn how to adapt your app to these response limits, see Background Location Limits.

The following snippet shows how to define a `BroadcastReceiver` that posts a notification when a geofence transition occurs. When the user clicks the notification, the app's main activity appears:

```
class GeofenceBroadcastReceiver : BroadcastReceiver() {
    // ...
    override fun onReceive(context: Context?, intent: Intent?) {
        val geofencingEvent = GeofencingEvent.fromIntent(intent)
        if (geofencingEvent.hasError()) {
            val errorMessage = GeofenceStatusCodes
                    .getStatusCodeString(geofencingEvent.errorCode)
            Log.e(TAG, errorMessage)
```

```
            return
        }

        // Get the transition type.
        val geofenceTransition = geofencingEvent.geofenceTransition

        // 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.
            val triggeringGeofences = geofencingEvent.triggeringGeofences

            // Get the transition details as a String.
            val 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))
        }
    }
}
```

```
public class GeofenceBroadcastReceiver extends BroadcastReceiver {
    // ...
    protected void onReceive(Context context, Intent intent) {
        GeofencingEvent geofencingEvent = GeofencingEvent.fromIntent(intent);
        if (geofencingEvent.hasError()) {
            String errorMessage = GeofenceStatusCodes
                    .getStatusCodeString(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));
        }
    }
}
```

After detecting the transition event via the `PendingIntent`, the `BroadcastReceiver` gets the geofence transition type and tests whether it is one of the events the app uses to trigger notifications -- either `GEOFENCE_TRANSITION_ENTER` or `GEOFENCE_TRANSITION_EXIT` in this case. The service then sends a notification and logs the transition details.

Stop geofence monitoring

Stopping geofence monitoring when it is no longer needed or desired can help save battery power and CPU cycles on the device. You can stop geofence monitoring in the main activity used to add and remove geofences; removing a geofence stops it immediately. The API provides methods to remove geofences either by request IDs, or by removing geofences associated with a given `PendingIntent`.

The following snippet removes geofences by `PendingIntent`, stopping all further notification when the device enters or exits previously added geofences:

```
geofencingClient?.removeGeofences(geofencePendingIntent)?.run {
    addOnSuccessListener {
        // Geofences removed
        // ...
    }
```

```
    addOnFailureListener {
        // Failed to remove geofences
        // ...
    }
}
```

```
geofencingClient.removeGeofences(getGeofencePendingIntent())
        .addOnSuccessListener(this, new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
                // Geofences removed
                // ...
            }
        })
        .addOnFailureListener(this, new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                // Failed to remove geofences
                // ...
            }
        });
```

You can combine geofencing with other location-aware features, such as periodic location updates. For more information, see the other lessons in this class.

Use best practices for geofencing

This section outlines recommendations for using geofencing with the location APIs for Android.

Reduce power consumption

You can use the following techniques to optimize power consumption in your apps that use geofencing:

- Set the notification responsiveness to a higher value. Doing so improves power consumption by increasing the latency of geofence alerts. For example, if you set a responsiveness value of five minutes your app only checks for an entrance or exit alert once every five minutes. Setting lower values doesn't necessarily mean that users are notified within that time period (for example, if you set a value of 5 seconds it may take a bit longer than that to receive the alert).
- Use a larger geofence radius for locations where a user spends a significant amount of time, such as home or work. While a larger radius doesn't directly reduce power consumption, it reduces the frequency at which the app checks for entrance or exit, effectively lowering overall power consumption.

Choose the optimal radius for your geofence

For best results, the minimum radius of the geofence should be set between 100 - 150 meters. When Wi-Fi is available location accuracy is usually between 20 - 50 meters. When indoor location is available, the accuracy range can be as small as 5 meters. Unless you know indoor location is available inside the geofence, assume that Wi-Fi location accuracy is about 50 meters.

When Wi-Fi location isn't available (for example, when you are driving in rural areas) the location accuracy degrades. The accuracy range can be as large as several hundred meters to several kilometers. In cases like this, you should create geofences using a larger radius.

Explain to users why your app uses geofencing

Because your app accesses location in the background when you use geofencing, consider how your app delivers benefits to users. Explain to them clearly why your app needs this access to increase user understanding and transparency.

For more information about best practices related to location access, including geofencing, see the privacy best practices page.

Use the dwell transition type to reduce alert spam

If you receive a large number of alerts when driving briefly past a geofence, the best way to reduce the alerts is to use a transition type of `GEOFENCE_TRANSITION_DWELL` instead of `GEOFENCE_TRANSITION_ENTER`. This way, the dwelling alert is sent only when the user stops inside a geofence for a given period of time. You can choose the duration by setting a loitering delay.

Re-register geofences only when required

Registered geofences are kept in the `com.google.process.location` process owned by the `com.google.android.gms` package. The app doesn't need to do anything to handle the following events, because the system restores geofences after these events:

- Google Play services is upgraded.
- Google Play services is killed and restarted by the system due resource restriction.
- The location process crashes.

The app must re-register geofences if they're still needed after the following events, since the system cannot recover the geofences in the following cases:

- The device is rebooted. The app should listen for the device's boot complete action, and then re- register the geofences required.
- The app is uninstalled and re-installed.
- The app's data is cleared.
- Google Play services data is cleared.
- The app has received a `GEOFENCE_NOT_AVAILABLE` alert. This typically happens after NLP (Android's Network Location Provider) is disabled.

Troubleshoot the geofence entrance event

If geofences aren't being triggered when the device enters a geofence (the `GEOFENCE_TRANSITION_ENTER` alert isn't triggered), first ensure that your geofences are registered properly as described in this guide.

Here are some possible reasons for alerts not working as expected:

- **Accurate location isn't available inside your geofence or your geofence is too small.** On most devices, the geofence service uses only network location for geofence triggering. The service uses this approach because network location consumes much less power, it takes less time to get discrete locations, and most importantly it's available indoors.
- **Wi-Fi is turned off on the device.** Having Wi-Fi on can significantly improve the location accuracy, so if Wi-Fi is turned off, your application might never get geofence alerts depending on several settings including the radius of the geofence, the device model, or the Android version. Starting from Android 4.3 (API level 18), we added the capability of "Wi-Fi scan only mode" which allows users to disable Wi-Fi but still get good network location. It's good practice to prompt the user and provide a shortcut for the user to enable Wi-Fi or Wi-Fi scan only mode if both of them are disabled. Use SettingsClient to ensure that the device's system settings are properly configured for optimal location detection.

  Note: If your app targets Android 10 (API level 29) or higher, you cannot call `WifiManager.setEnabled()` directly unless your app is a system app or a device policy controller (DPC). Instead, use a settings panel.
- **There is no reliable network connectivity inside your geofence.** If there is no reliable data connection, alerts might not be generated. This is because the geofence service depends on the network location provider which in turn requires a data connection.
- **Alerts can be late.** The geofence service doesn't continuously query for location, so expect some latency when receiving alerts. Usually the latency is less than 2 minutes, even less when the device has been moving. If Background Location Limits are in effect, the latency is about 2-3 minutes on average. If the device has been stationary for a significant period of time, the latency may increase (up to 6 minutes).

Additional resources

To learn more about Geofencing, view the following materials:

Samples

Sample app for creating and monitoring geofences.
