Creating and Monitoring Geofences

Geofencing combines awareness of the user's current location with awareness of nearby features, defined as 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. You can have multiple active geofences at one time.

Location Services treats a geofences as an area rather than as a points and proximity. This allows it to detect when the user enters or exits a geofence. For each geofence, you can ask Location Services to send you entrance events or exit events or both. You can also limit the duration of a geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.

THIS LESSON TEACHES YOU TO

1. Request Geofence Monitoring
2. Handle Geofence Transitions
3. Stop Geofence Monitoring

YOU SHOULD ALSO READ

- Setup Google Play Services SDK

TRY IT OUT

Download the sample

GeofenceDetection.zip

Request Geofence Monitoring

The first step in requesting geofence monitoring is to request the necessary permission. To use geofencing, your app must request `ACCESS_FINE_LOCATION` (/reference/android/Manifest.permission.html#ACCESS_FINE_LOCATION). To request this permission, add the following element as a child element of the `<manifest>` (/guide/topics/manifest/manifest-element.html) element:

```
<uses-permission android:name="android.permission.ACCESS_FINE_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:

```
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("Geofence Detection",
                    "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(),
                        "Geofence Detection");
            }
        }
    }
    ...
}
```

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

To use geofencing, start by defining the geofences you want to monitor. Although you usually store geofence data in a local database or download it from the network, you need to send a geofence to Location Services as an instance of `Geofence` (/reference/com/google/android/gms/location/Geofence.html), which you create with `Geofence.Builder` (/reference/com/google/android/gms/location/Geofence.Builder.html). Each `Geofence` (/reference/com/google/android/gms/location/Geofence.html) object contains the following information:

Latitude, longitude, and radius
: Define a circular area for the geofence. Use the latitude and longitude to mark a location of interest, and then use the radius to adjust how close the user needs to approach the location before the geofence is detected. The larger the radius, the more likely the user will trigger a geofence transition alert by approaching the geofence. For example, providing a large radius for a geofencing app that turns on lights in the user's house as the user returns home might cause the lights to go on even if the user is simply passing by.

Expiration time
: How long the geofence should remain active. Once the expiration time is reached, Location Services

deletes the geofence. Most of the time, you should specify an expiration time, but you may want to keep permanent geofences for the user's home or place of work.

Transition type
: Location Services can detect when the user steps within the radius of the geofence ("entry") and when the user steps outside the radius of the geofence ("exit"), or both.

Geofence ID
: A string that is stored with the geofence. You should make this unique, so that you can use it to remove a geofence from Location Services tracking.

Define geofence storage

A geofencing app needs to read and write geofence data to persistent storage. You shouldn't use `Geofence` (/reference/com/google/android/gms/location/Geofence.html) objects to do this; instead, use storage techniques such as databases that can store groups of related data.

As an example of storing geofence data, the following snippet defines two classes that use the app's `SharedPreferences` (/reference/android/content/SharedPreferences.html) instance for persistent storage. The class `SimpleGeofence`, analogous to a database record, stores the data for a single `Geofence` (/reference/com/google/android/gms/location/Geofence.html) object in a "flattened" form. The class `SimpleGeofenceStore`, analogous to a database, reads and writes `SimpleGeofence` data to the `SharedPreferences` (/reference/android/content/SharedPreferences.html) instance.

```
public class MainActivity extends FragmentActivity {
    ...
    /**
     * A single Geofence object, defined by its center and radius.
     */
    public class SimpleGeofence {
            // Instance variables
            private final String mId;
            private final double mLatitude;
            private final double mLongitude;
            private final float mRadius;
            private long mExpirationDuration;
            private int mTransitionType;

        /**
         * @param geofenceId The Geofence's request ID
         * @param latitude Latitude of the Geofence's center.
         * @param longitude Longitude of the Geofence's center.
         * @param radius Radius of the geofence circle.
         * @param expiration Geofence expiration duration
         * @param transition Type of Geofence transition.
         */
        public SimpleGeofence(
                String geofenceId,
                double latitude,
                double longitude,
                float radius,
                long expiration,
                int transition) {
            // Set the instance fields from the constructor
            this.mId = geofenceId;
            this.mLatitude = latitude;
            this.mLongitude = longitude;
            this.mRadius = radius;
            this.mExpirationDuration = expiration;
            this.mTransitionType = transition;
        }
        // Instance field getters
        public String getId() {
```

```
            return mId;
        }
        public double getLatitude() {
            return mLatitude;
        }
        public double getLongitude() {
            return mLongitude;
        }
        public float getRadius() {
            return mRadius;
        }
        public long getExpirationDuration() {
            return mExpirationDuration;
        }
        public int getTransitionType() {
            return mTransitionType;
        }
        /**
         * Creates a Location Services Geofence object from a
         * SimpleGeofence.
         *
         * @return A Geofence object
         */
        public Geofence toGeofence() {
            // Build a new Geofence object
            return new Geofence.Builder()
                    .setRequestId(getId())
                    .setTransitionTypes(mTransitionType)
                    .setCircularRegion(
                            getLatitude(), getLongitude(), getRadius())
                    .setExpirationDuration(mExpirationDuration)
                    .build();
        }
    }
    ...
    /**
     * Storage for geofence values, implemented in SharedPreferences.
     */
    public class SimpleGeofenceStore {
        // Keys for flattened geofences stored in SharedPreferences
        public static final String KEY_LATITUDE =
                "com.example.android.geofence.KEY_LATITUDE";
        public static final String KEY_LONGITUDE =
                "com.example.android.geofence.KEY_LONGITUDE";
        public static final String KEY_RADIUS =
                "com.example.android.geofence.KEY_RADIUS";
        public static final String KEY_EXPIRATION_DURATION =
                "com.example.android.geofence.KEY_EXPIRATION_DURATION";
        public static final String KEY_TRANSITION_TYPE =
                "com.example.android.geofence.KEY_TRANSITION_TYPE";
        // The prefix for flattened geofence keys
        public static final String KEY_PREFIX =
                "com.example.android.geofence.KEY";
        /*
         * Invalid values, used to test geofence storage when
         * retrieving geofences
         */
        public static final long INVALID_LONG_VALUE = -999l;
        public static final float INVALID_FLOAT_VALUE = -999.0f;
        public static final int INVALID_INT_VALUE = -999;
        // The SharedPreferences object in which geofences are stored
```

```
private final SharedPreferences mPrefs;
// The name of the SharedPreferences
private static final String SHARED_PREFERENCES =
        "SharedPreferences";
// Create the SharedPreferences storage with private access only
public SimpleGeofenceStore(Context context) {
    mPrefs =
            context.getSharedPreferences(
                    SHARED_PREFERENCES,
                    Context.MODE_PRIVATE);
}
/**
 * Returns a stored geofence by its id, or returns null
 * if it's not found.
 *
 * @param id The ID of a stored geofence
 * @return A geofence defined by its center and radius. See
 */
public SimpleGeofence getGeofence(String id) {
    /*
     * Get the latitude for the geofence identified by id, or
     * INVALID_FLOAT_VALUE if it doesn't exist
     */
    double lat = mPrefs.getFloat(
            getGeofenceFieldKey(id, KEY_LATITUDE),
            INVALID_FLOAT_VALUE);
    /*
     * Get the longitude for the geofence identified by id, or
     * INVALID_FLOAT_VALUE if it doesn't exist
     */
    double lng = mPrefs.getFloat(
            getGeofenceFieldKey(id, KEY_LONGITUDE),
            INVALID_FLOAT_VALUE);
    /*
     * Get the radius for the geofence identified by id, or
     * INVALID_FLOAT_VALUE if it doesn't exist
     */
    float radius = mPrefs.getFloat(
            getGeofenceFieldKey(id, KEY_RADIUS),
            INVALID_FLOAT_VALUE);
    /*
     * Get the expiration duration for the geofence identified
     * by id, or INVALID_LONG_VALUE if it doesn't exist
     */
    long expirationDuration = mPrefs.getLong(
            getGeofenceFieldKey(id, KEY_EXPIRATION_DURATION),
            INVALID_LONG_VALUE);
    /*
     * Get the transition type for the geofence identified by
     * id, or INVALID_INT_VALUE if it doesn't exist
     */
    int transitionType = mPrefs.getInt(
            getGeofenceFieldKey(id, KEY_TRANSITION_TYPE),
            INVALID_INT_VALUE);
    // If none of the values is incorrect, return the object
    if (
        lat != GeofenceUtils.INVALID_FLOAT_VALUE &&
        lng != GeofenceUtils.INVALID_FLOAT_VALUE &&
        radius != GeofenceUtils.INVALID_FLOAT_VALUE &&
        expirationDuration !=
                GeofenceUtils.INVALID_LONG_VALUE &&
```

```
            transitionType != GeofenceUtils.INVALID_INT_VALUE) {

            // Return a true Geofence object
            return new SimpleGeofence(
                    id, lat, lng, radius, expirationDuration,
                    transitionType);
        // Otherwise, return null.
        } else {
            return null;
        }
    }
    /**
     * Save a geofence.
     * @param geofence The SimpleGeofence containing the
     * values you want to save in SharedPreferences
     */
    public void setGeofence(String id, SimpleGeofence geofence) {
        /*
         * Get a SharedPreferences editor instance. Among other
         * things, SharedPreferences ensures that updates are atomic
         * and non-concurrent
         */
        Editor editor = mPrefs.edit();
        // Write the Geofence values to SharedPreferences
        editor.putFloat(
                getGeofenceFieldKey(id, KEY_LATITUDE),
                (float) geofence.getLatitude());
        editor.putFloat(
                getGeofenceFieldKey(id, KEY_LONGITUDE),
                (float) geofence.getLongitude());
        editor.putFloat(
                getGeofenceFieldKey(id, KEY_RADIUS),
                geofence.getRadius());
        editor.putLong(
                getGeofenceFieldKey(id, KEY_EXPIRATION_DURATION),
                geofence.getExpirationDuration());
        editor.putInt(
                getGeofenceFieldKey(id, KEY_TRANSITION_TYPE),
                geofence.getTransitionType());
        // Commit the changes
        editor.commit();
    }
    public void clearGeofence(String id) {
        /*
         * Remove a flattened geofence object from storage by
         * removing all of its keys
         */
        Editor editor = mPrefs.edit();
        editor.remove(getGeofenceFieldKey(id, KEY_LATITUDE));
        editor.remove(getGeofenceFieldKey(id, KEY_LONGITUDE));
        editor.remove(getGeofenceFieldKey(id, KEY_RADIUS));
        editor.remove(getGeofenceFieldKey(id,
                KEY_EXPIRATION_DURATION));
        editor.remove(getGeofenceFieldKey(id, KEY_TRANSITION_TYPE));
        editor.commit();
    }
    /**
     * Given a Geofence object's ID and the name of a field
     * (for example, KEY_LATITUDE), return the key name of the
     * object's values in SharedPreferences.
     *
```

```
         * @param id The ID of a Geofence object
         * @param fieldName The field represented by the key
         * @return The full key name of a value in SharedPreferences
         */
        private String getGeofenceFieldKey(String id,
                String fieldName) {
            return KEY_PREFIX + "_" + id + "_" + fieldName;
        }
    }
    ...
}
```

Create Geofence objects

The following snippet uses the `SimpleGeofence` and `SimpleGeofenceStore` classes gets geofence data from the UI, stores it in `SimpleGeofence` objects, stores these objects in a `SimpleGeofenceStore` object, and then creates Geofence (/reference/com/google/android/gms/location/Geofence.html) objects:

```
public class MainActivity extends FragmentActivity {
    ...
    /*
     * Use to set an expiration time for a geofence. After this amount
     * of time Location Services will stop tracking the geofence.
     */
    private static final long SECONDS_PER_HOUR = 60;
    private static final long MILLISECONDS_PER_SECOND = 1000;
    private static final long GEOFENCE_EXPIRATION_IN_HOURS = 12;
    private static final long GEOFENCE_EXPIRATION_TIME =
            GEOFENCE_EXPIRATION_IN_HOURS *
            SECONDS_PER_HOUR *
            MILLISECONDS_PER_SECOND;
    ...
    /*
     * Handles to UI views containing geofence data
     */
    // Handle to geofence 1 latitude in the UI
    private EditText mLatitude1;
    // Handle to geofence 1 longitude in the UI
    private EditText mLongitude1;
    // Handle to geofence 1 radius in the UI
    private EditText mRadius1;
    // Handle to geofence 2 latitude in the UI
    private EditText mLatitude2;
    // Handle to geofence 2 longitude in the UI
    private EditText mLongitude2;
    // Handle to geofence 2 radius in the UI
    private EditText mRadius2;
    /*
     * Internal geofence objects for geofence 1 and 2
     */
    private SimpleGeofence mUIGeofence1;
    private SimpleGeofence mUIGeofence2;
    ...
    // Internal List of Geofence objects
    List<Geofence> mGeofenceList;
    // Persistent storage for geofences
    private SimpleGeofenceStore mGeofenceStorage;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        ...
        // Instantiate a new geofence storage area
        mGeofenceStorage = new SimpleGeofenceStore(this);

        // Instantiate the current List of geofences
        mCurrentGeofences = new ArrayList<Geofence>();
    }
    ...
    /**
     * Get the geofence parameters for each geofence from the UI
     * and add them to a List.
     */
    public void createGeofences() {
        /*
         * Create an internal object to store the data. Set its
         * ID to "1". This is a "flattened" object that contains
         * a set of strings
         */
        mUIGeofence1 = new SimpleGeofence(
                "1",
                Double.valueOf(mLatitude1.getText().toString()),
                Double.valueOf(mLongitude1.getText().toString()),
                Float.valueOf(mRadius1.getText().toString()),
                GEOFENCE_EXPIRATION_TIME,
                // This geofence records only entry transitions
                Geofence.GEOFENCE_TRANSITION_ENTER);
        // Store this flat version
        mGeofenceStorage.setGeofence("1", mUIGeofence1);
        // Create another internal object. Set its ID to "2"
        mUIGeofence2 = new SimpleGeofence(
                "2",
                Double.valueOf(mLatitude2.getText().toString()),
                Double.valueOf(mLongitude2.getText().toString()),
                Float.valueOf(mRadius2.getText().toString()),
                GEOFENCE_EXPIRATION_TIME,
                // This geofence records both entry and exit transitions
                Geofence.GEOFENCE_TRANSITION_ENTER |
                Geofence.GEOFENCE_TRANSITION_EXIT);
        // Store this flat version
        mGeofenceStorage.setGeofence(2, mUIGeofence2);
        mGeofenceList.add(mUIGeofence1.toGeofence());
        mGeofenceList.add(mUIGeofence2.toGeofence());
    }
    ...
}
```

In addition to the List (/reference/java/util/List.html) of Geofence (/reference/com/google/android/gms/location/Geofence.html) objects you want to monitor, you need to provide Location Services with the Intent (/reference/android/content/Intent.html) that it sends to your app when it detects geofence transitions.

Define a Intent for geofence transitions

The Intent (/reference/android/content/Intent.html) 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, an IntentService (/reference/android/app/IntentService.html) is a good way to handle the intent. An IntentService (/reference/android/app/IntentService.html) can post a notification, do long-running background work, send intents to other services, or send a broadcast intent. The following snippet shows how how to define a

PendingIntent (/reference/android/app/PendingIntent.html) that starts an IntentService (/reference/android/app/IntentService.html):

```
public class MainActivity extends FragmentActivity {
    ...
    /*
     * Create a PendingIntent that triggers an IntentService in your
     * app when a geofence transition occurs.
     */
    private PendingIntent getTransitionPendingIntent() {
        // Create an explicit Intent
        Intent intent = new Intent(this,
                ReceiveTransitionsIntentService.class);
        /*
         * Return the PendingIntent
         */
        return PendingIntent.getService(
                this,
                0,
                intent,
                PendingIntent.FLAG_UPDATE_CURRENT);
    }
    ...
}
```

Now you have all the code you need to send a request to monitor geofences to Location Services.

Send the monitoring request

Sending the monitoring request requires two asynchronous operations. The first operation gets a location client for the request, and the second makes the request using the client. In both cases, Location Services invokes a callback method when it finishes the operation. The best way to handle these operations is to chain together the method calls. The following snippets demonstrate how to set up an activity, define the methods, and call them in the proper order.

First, modify the activity's class definition to implement the necessary callback interfaces. Add the following interfaces:

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.

OnAddGeofencesResultListener
: Specifies a method that Location Services calls once it has added the geofences.

For example:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
}
```

Start the request process

Next, define a method that starts the request process by connecting to Location Services. Mark this as a request to add a geofence by setting a global variable. This allows you to use the callback

ConnectionCallbacks.onConnected()
(/reference/com/google/android/gms/common/GooglePlayServicesClient.ConnectionCallbacks.html#onConnected(android.os.Bundle)) to add geofences and to remove them, as described in succeeding sections.

To guard against race conditions that might arise if your app tries to start another request before the first one finishes, define a boolean flag that tracks the state of the current request:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
    // Holds the location client
    private LocationClient mLocationClient;
    // Stores the PendingIntent used to request geofence monitoring
    private PendingIntent mGeofenceRequestIntent;
    // Defines the allowable request types.
    public enum REQUEST_TYPE = {ADD}
    private REQUEST_TYPE mRequestType;
    // Flag that indicates if a request is underway.
    private boolean mInProgress;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // Start with the request flag set to false
        mInProgress = false;
        ...
    }
    ...
    /**
     * Start a request for geofence monitoring by calling
     * LocationClient.connect().
     */
    public void addGeofences() {
        // Start a request to add geofences
        mRequestType = ADD;
        /*
         * Test for Google Play services after setting the request type.
         * If Google Play services isn't present, the proper request
         * can be restarted.
         */
        if (!servicesConnected()) {
            return;
        }
        /*
         * Create a new location client object. Since the current
         * activity class implements ConnectionCallbacks and
         * OnConnectionFailedListener, pass the current activity object
         * as the listener for both parameters
         */
        mLocationClient = new LocationClient(this, this, this)
        // If a request is not already underway
        if (!mInProgress) {
            // Indicate that a request is underway
            mInProgress = true;
            // Request a connection from the client to Location Services
            mLocationClient.connect();
        } else {
            /*
             * A request is already underway. You can handle
```

```
                * this situation by disconnecting the client,
                * re-setting the flag, and then re-trying the
                * request.
                */
            }
        }
        ...
}
```

Send a request to add the geofences

In your implementation of [ConnectionCallbacks.onConnected()](/reference/com/google/android/gms/common/GooglePlayServicesClient.ConnectionCallbacks.html#onConnected(android.os.Bundle)), call [LocationClient.addGeofences()](/reference/com/google/android/gms/location/LocationClient.html#addGeofences(java.util.List<com.google.android.gms.location.Geofence>, android.app.PendingIntent, com.google.android.gms.location.LocationClient.OnAddGeofencesResultListener)). Notice that if the connection fails, [onConnected()](/reference/com/google/android/gms/common/GooglePlayServicesClient.ConnectionCallbacks.html#onConnected(android.os.Bundle)) isn't called, and the request stops.

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
    /*
     * Provide the implementation of ConnectionCallbacks.onConnected()
     * Once the connection is available, send a request to add the
     * Geofences
     */
    @Override
    private void onConnected(Bundle dataBundle) {
        ...
        switch (mRequestType) {
            case ADD :
                // Get the PendingIntent for the request
                mTransitionPendingIntent =
                        getTransitionPendingIntent();
                // Send a request to add the current geofences
                mLocationClient.addGeofences(
                        mCurrentGeofences, pendingIntent, this);
            ...
        }
    }
    ...
}
```

Notice that [addGeofences()](/reference/com/google/android/gms/location/LocationClient.html#addGeofences(java.util.List<com.google.android.gms.location.Geofence>, android.app.PendingIntent, com.google.android.gms.location.LocationClient.OnAddGeofencesResultListener)) returns immediately, but the status of the request is indeterminate until Location Services calls [onAddGeofencesResult()](/reference/com/google/android/gms/location/LocationClient.OnAddGeofencesResultListener.html#onAddGeofencesResult(int, java.lang.String[])) Once this method is called, you can determine if the request was successful or not.

Check the result returned by Location Services

When Location Services invokes your implementation of the callback method onAddGeofencesResult() (/reference/com/google/android/gms/location/LocationClient.OnAddGeofencesResultListener.html#onAddGeofencesResult(int, java.lang.String[])), indicating that the request is complete, examine the incoming status code. If the request was successful, the geofences you requested are active. If the request was unsuccessful, the geofences aren't active, and you need to re-try the request or report an error. For example:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
        ...
    /*
     * Provide the implementation of
     * OnAddGeofencesResultListener.onAddGeofencesResult.
     * Handle the result of adding the geofences
     *
     */
    @Override
    public void onAddGeofencesResult(
            int statusCode, String[] geofenceRequestIds) {
        // If adding the geofences was successful
        if (LocationStatusCodes.SUCCESS == statusCode) {
            /*
             * Handle successful addition of geofences here.
             * You can send out a broadcast intent or update the UI.
             * geofences into the Intent's extended data.
             */
        } else {
        // If adding the geofences failed
            /*
             * Report errors here.
             * You can log the error using Log.e() or update
             * the UI.
             */
        }
        // Turn off the in progress flag and disconnect the client
        mInProgress = false;
        mLocationClient.disconnect();
    }
    ...
}
```

Handle disconnections

In some cases, Location Services may disconnect from the activity recognition client before you call disconnect() (/reference/com/google/android/gms/location/LocationClient.html#disconnect()). To handle this situation, implement onDisconnected() (/reference/com/google/android/gms/common/GooglePlayServicesClient.ConnectionCallbacks.html#onDisconnected()). In this method, set the request flag to indicate that a request is not in progress, and delete the client:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
    /*
     * Implement ConnectionCallbacks.onDisconnected()
     * Called by Location Services once the location client is
     * disconnected.
```

```
     */
    @Override
    public void onDisconnected() {
        // Turn off the request flag
        mInProgress = false;
        // Destroy the current location client
        mLocationClient = null;
    }
    ...
}
```

Handle connection errors

Besides handling the normal callbacks from Location Services, you have to provide a callback method that Location Services calls if a connection error occurs. This callback method can re-use the DialogFragment (/reference/android/support/v4/app/DialogFragment.html) class that you defined to handle the check for Google Play services. It can also re-use the override you defined for onActivityResult() (/reference/android/support/v4/app/FragmentActivity.html#onActivityResult(int, int, android.content.Intent)) that receives any Google Play services results that occur when the user interacts with the error dialog. The following snippet shows you a sample implementation of the callback method:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
    // Implementation of OnConnectionFailedListener.onConnectionFailed
    @Override
    public void onConnectionFailed(ConnectionResult connectionResult) {
        // Turn off the request flag
        mInProgress = false;
        /*
         * If the error has a resolution, start a Google Play services
         * activity to resolve it.
         */
        if (connectionResult.hasResolution()) {
            try {
                connectionResult.startResolutionForResult(
                        this,
                        CONNECTION_FAILURE_RESOLUTION_REQUEST);
            } catch (SendIntentException e) {
                // Log the error
                e.printStackTrace();
            }
        // If no resolution is available, display an error dialog
        } 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(),
                        "Geofence Detection");
            }
        }
    }
    ...
}
```

Handle Geofence Transitions

When Location Services detects that the user has entered or exited a geofence, it sends out the Intent (/reference/android/content/Intent.html) contained in the PendingIntent (/reference/android/app/PendingIntent.html) you included in the request to add geofences. This Intent (/reference/android/content/Intent.html) is

Define an IntentService

The following snippet shows how to define an IntentService (/reference/android/app/IntentService.html) that posts a notification when a geofence transition occurs. When the user clicks the notification, the app's main activity appears:

```
public class ReceiveTransitionsIntentService extends IntentService {
    ...
    /**
     * Sets an identifier for the service
     */
    public ReceiveTransitionsIntentService() {
        super("ReceiveTransitionsIntentService");
    }
    /**
     * Handles incoming intents
     *@param intent The Intent sent by Location Services. This
     * Intent is provided
     * to Location Services (inside a PendingIntent) when you call
     * addGeofences()
     */
    @Override
    protected void onHandleIntent(Intent intent) {
        // First check for errors
        if (LocationClient.hasError(intent)) {
            // Get the error code with a static method
            int errorCode = LocationClient.getErrorCode(intent);
            // Log the error
            Log.e("ReceiveTransitionsIntentService",
                    "Location Services error: " +
                    Integer.toString(errorCode));
            /*
             * You can also send the error code to an Activity or
             * Fragment with a broadcast Intent
             */
        /*
         * If there's no error, get the transition type and the IDs
         * of the geofence or geofences that triggered the transition
         */
        } else {
            // Get the type of transition (entry or exit)
```

```
            int transitionType =
                    LocationClient.getGeofenceTransition(intent);
            // Test that a valid transition was reported
            if (
                (transitionType == Geofence.GEOFENCE_TRANSITION_ENTER)
                 ||
                (transitionType == Geofence.GEOFENCE_TRANSITION_EXIT)
               ) {
                List <Geofence> triggerList =
                        getTriggeringGeofences(intent);

                String[] triggerIds = new String[geofenceList.size()];

                for (int i = 0; i < triggerIds.length; i++) {
                    // Store the Id of each geofence
                    triggerIds[i] = triggerList.get(i).getRequestId();
                }
                /*
                 * At this point, you can store the IDs for further use
                 * display them, or display the details associated with
                 * them.
                 */
            }
        // An invalid transition was reported
        } else {
            Log.e("ReceiveTransitionsIntentService",
                    "Geofence transition error: " +
                    Integer.toString()transitionType));
        }
    }
    ...
}
```

Specify the IntentService in the manifest

To identify the `IntentService` (/reference/android/app/IntentService.html) to the system, add a `<service>` (/guide/topics/manifest/service-element.html) element to the app manifest. For example:

```
<service
    android:name="com.example.android.location.ReceiveTransitionsIntentService"
    android:label="@string/app_name"
    android:exported="false">
</service>
```

Notice that you don't have to specify intent filters for the service, because it only receives explicit intents. How the incoming geofence transition intents are created is described in the section Send the monitoring request (#requestmonitoring).

Stop Geofence Monitoring

To stop geofence monitoring, you remove the geofences themselves. You can remove a specific set of geofences or all the geofences associated with a `PendingIntent` (/reference/android/app/PendingIntent.html). The procedure is similar to adding geofences. The first operation gets a location client for the removal request, and the second makes the request using the client.

The callback methods that Location Services invokes when it has finished removing geofences are defined in the interface `LocationClient.OnRemoveGeofencesResultListener` (/reference/com/google/android/gms/location/LocationClient.OnRemoveGeofencesResultListener.html).

Declare this interface as part of your class definition, and then add definitions for its two methods:

`onRemoveGeofencesByPendingIntentResult()`
: Callback invoked when Location Services finishes a request to remove all geofences made by the method `removeGeofences(PendingIntent, LocationClient.OnRemoveGeofencesResultListener)`.

`onRemoveGeofencesByRequestIdsResult(List<String>, LocationClient.OnRemoveGeofencesResultListener)`
: Callback invoked when Location Services finished a request to remove a set of geofences, specified by their geofence IDs, by the method `removeGeofences(List<String>, LocationClient.OnRemoveGeofencesResultListener)`.

Examples of implementing these methods are shown in the next snippets.

Remove all geofences

Since removing geofences uses some of the methods you use to add geofences, start by defining another request type:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
    // Enum type for controlling the type of removal requested
    public enum REQUEST_TYPE = {ADD, REMOVE_INTENT}
    ...
}
```

Start the removal request by getting a connection to Location Services. If the connection fails, `onConnected()` (/reference/com/google/android/gms/common/GooglePlayServicesClient.ConnectionCallbacks.html#onConnected(android.os.Bundle)) isn't called, and the request stops. The following snippet shows how to start the request:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
    /**
     * Start a request to remove geofences by calling
     * LocationClient.connect()
     */
    public void removeGeofences(PendingIntent requestIntent) {
        // Record the type of removal request
        mRequestType = REMOVE_INTENT;
        /*
         * Test for Google Play services after setting the request type.
         * If Google Play services isn't present, the request can be
         * restarted.
         */
        if (!servicesConnected()) {
            return;
        }
        // Store the PendingIntent
        mGeofenceRequestIntent = requestIntent;
        /*
         * Create a new location client object. Since the current
         * activity class implements ConnectionCallbacks and
         * OnConnectionFailedListener, pass the current activity object
         * as the listener for both parameters
```

```
             */
            mLocationClient = new LocationClient(this, this, this);
            // If a request is not already underway
            if (!mInProgress) {
                // Indicate that a request is underway
                mInProgress = true;
                // Request a connection from the client to Location Services
                mLocationClient.connect();
            } else {
                /*
                 * A request is already underway. You can handle
                 * this situation by disconnecting the client,
                 * re-setting the flag, and then re-trying the
                 * request.
                 */
            }
        }
        ...
}
```

When Location Services invokes the callback method indicating that the connection is open, make the request to remove all geofences. Disconnect the client after making the request. For example:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
    /**
     * Once the connection is available, send a request to remove the
     * Geofences. The method signature used depends on which type of
     * remove request was originally received.
     */
    private void onConnected(Bundle dataBundle) {
        /*
         * Choose what to do based on the request type set in
         * removeGeofences
         */
        switch (mRequestType) {
            ...
            case REMOVE_INTENT :
                mLocationClient.removeGeofences(
                        mGeofenceRequestIntent, this);
                break;
            ...
        }
    }
    ...
}
```

Although the call to removeGeofences(PendingIntent, LocationClient.OnRemoveGeofencesResultListener) (/reference/com/google/android/gms/location/LocationClient.html#removeGeofences(android.app.PendingIntent, com.google.android.gms.location.LocationClient.OnRemoveGeofencesResultListener)) Services calls returns immediately, the result of the removal request is indeterminate until Location Services calls onRemoveGeofencesByPendingIntentResult() (/reference/com/google/android/gms/location/LocationClient.OnRemoveGeofencesResultListener.html#onRemoveGeofencesByPendingIntentResult(int, android.app.PendingIntent)). The following snippet shows how to define this method:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
    /**
     * When the request to remove geofences by PendingIntent returns,
     * handle the result.
     *
     *@param statusCode the code returned by Location Services
     *@param requestIntent The Intent used to request the removal.
     */
    @Override
    public void onRemoveGeofencesByPendingIntentResult(int statusCode,
            PendingIntent requestIntent) {
        // If removing the geofences was successful
        if (statusCode == LocationStatusCodes.SUCCESS) {
            /*
             * Handle successful removal of geofences here.
             * You can send out a broadcast intent or update the UI.
             * geofences into the Intent's extended data.
             */
        } else {
        // If adding the geocodes failed
            /*
             * Report errors here.
             * You can log the error using Log.e() or update
             * the UI.
             */
        }
        /*
         * Disconnect the location client regardless of the
         * request status, and indicate that a request is no
         * longer in progress
         */
        mInProgress = false;
        mLocationClient.disconnect();
    }
    ...
}
```

Remove individual geofences

The procedure for removing an individual geofence or set of geofences is similar to the removal of all geofences. To specify the geofences you want remove, add their geofence ID values to a `List (/reference/java/util/List.html)` of String objects. Pass this `List (/reference/java/util/List.html)` to a different definition of `removeGeofences` with the appropriate signature. This method then starts the removal process.

Start by adding a request type for removing geofences by a list, and also add a global variable for storing the list of geofences:

```
    ...
    // Enum type for controlling the type of removal requested
    public enum REQUEST_TYPE = {ADD, REMOVE_INTENT, REMOVE_LIST}
    // Store the list of geofence Ids to remove
    String<List> mGeofencesToRemove;
```

Next, define a list of geofences you want to remove. For example, this snippet removes the `Geofence`

(/reference/com/google/android/gms/location/Geofence.html) defined by the geofence ID "1":

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
        List<String> listOfGeofences =
                Collections.singletonList("1");
        removeGeofences(listOfGeofences);
    ...
}
```

The following snippet defines the `removeGeofences()` method:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
    /**
     * Start a request to remove monitoring by
     * calling LocationClient.connect()
     *
     */
    public void removeGeofences(List<String> geofenceIds) {
        // If Google Play services is unavailable, exit
        // Record the type of removal request
        mRequestType = REMOVE_LIST;
        /*
         * Test for Google Play services after setting the request type.
         * If Google Play services isn't present, the request can be
         * restarted.
         */
        if (!servicesConnected()) {
            return;
        }
        // Store the list of geofences to remove
        mGeofencesToRemove = geofenceIds;
        /*
         * Create a new location client object. Since the current
         * activity class implements ConnectionCallbacks and
         * OnConnectionFailedListener, pass the current activity object
         * as the listener for both parameters
         */
        mLocationClient = new LocationClient(this, this, this);
        // If a request is not already underway
        if (!mInProgress) {
            // Indicate that a request is underway
            mInProgress = true;
            // Request a connection from the client to Location Services
            mLocationClient.connect();
        } else {
            /*
             * A request is already underway. You can handle
             * this situation by disconnecting the client,
             * re-setting the flag, and then re-trying the
             * request.
             */
```

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

When Location Services invokes the callback method indicating that the connection is open, make the request to remove the list of geofences. Disconnect the client after making the request. For example:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
    private void onConnected(Bundle dataBundle) {
        ...
        switch (mRequestType) {
        ...
        // If removeGeofencesById was called
            case REMOVE_LIST :
                mLocationClient.removeGeofences(
                        mGeofencesToRemove, this);
                break;
        ...
        }
        ...
    }
    ...
}
```

Define an implementation of onRemoveGeofencesByRequestIdsResult() (/reference/com/google/android/gms/location/LocationClient.OnRemoveGeofencesResultListener.html#onRemoveGeofencesByRequestIdsResult(int, java.lang.String[])). Location Services invokes this callback method to indicate that the request to remove a list of geofences is complete. In this method, examine the incoming status code and take the appropriate action:

```
public class MainActivity extends FragmentAct     :y implements
        ConnectionCallbacks,
        OnConnectionFailedListener,
        OnAddGeofencesResultListener {
    ...
    /**
     * When the request to remove geofences h··  TDs returns, handle the
     * result.
     *
     * @param statusCo                          :ation Services
     * @param geofence
     */
    @Override
    public void onRemo                          ılt(
            int status                          iestIds) {
        // If removing                          ıl
        if (LocationSt                          :Code) {
            /*
             * Handle                           ences here.
             * You car                          ıt or update the UI.
             * geofenc                          ded data.
             */
        } else {
        // If removing
```

```
            /*
             * Report
             * You can log the error using Log.e() or update
             * the UI.
             */
        }
        // Indicate that a request is no longer in progress
        mInProgress = false;
        // Disconnect the location client
        mLocationClient.disconnect();
    }
    ...
}
```


You can combine geofencing with other location-aware features, such as periodic location updates or activity recognition, which are described in other lessons in this class.

The next lesson, Recognizing the User's Current Activity (activity-recognition.html), shows you how to request and receive activity updates. At regular intervals, Location Services can send you information about the user's current physical activity. Based on this information, you can change your app's behavior; for example, you can switch to a longer update interval if you detect that the user is walking instead of driving.