Recognizing the User's Current Activity

THIS LESSON TEACHES YOU TO

1. Request Activity Recognition Updates
2. Handle Activity Updates
3. Stop Activity Recognition Updates

YOU SHOULD ALSO READ

- Setup Google Play Services SDK
- Receiving Location Updates

TRY IT OUT

Download the sample

ActivityRecognition.zip

This lesson shows you how to request activity recognition updates from Location Services. Activity recognition tries to detect the user's current physical activity, such as walking, driving, or standing still. Requests for updates go through an activity recognition client, which, while different from the location client used by location or geofencing, follows a similar pattern. Based on the update interval you choose, Location Services sends out activity information containing one or more possible activities and the confidence level for each one.

Request Activity Recognition Updates

Requesting activity recognition updates from Location Services is similar to requesting periodic location updates. You send the request through a client, and Location Services sends updates back to your app by means of a `PendingIntent` `(/reference/android/app/PendingIntent.html)`. However, you need to request a special permission before you request activity updates, and you use a different type of client to make requests. The following sections show how to request the permission, connect the client, and request updates.

Request permission to receive updates

An app that wants to get activity recognition updates must have the permission `com.google.android.gms.permission.ACTIVITY_RECOGNITION`. To request this permission for your app, add the following XML element to your manifest as a child element of the `<manifest>` `(/guide/topics/manifest/manifest-element.html)` element:

```
<uses-permission
    android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION"/>
```

Activity recognition does not require the permissions `ACCESS_COARSE_LOCATION` `(/reference/android/Manifest.permission.html#ACCESS_COARSE_LOCATION)` or `ACCESS_FINE_LOCATION` `(/reference/android/Manifest.permission.html#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("Activity Recognition",
                    "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(),
                        "Activity Recognition");
            }
            return false;
        }
    }
    ...
}
```

Getting Started

Building Apps with
Multimedia

Building Apps with
Graphics & Animation

Building Apps with
Connectivity & the Cloud

Building Apps with
User Info & Location

Accessing Contacts Data

Remembering Users

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

Send the activity update request

Send the update request from an Activity (/reference/android/app/Activity.html) or Fragment (/reference/android/support/v4/app/Fragment.html) that implements the callback methods required by Location Servi hronous process that starts when you request a connection to an activity reco nnected, Location Services invokes your implementation of onConnected((/reference/c oglePlayServicesClient.ConnectionCallbacks.html#onConnected(android.o send the update request to Location Services; this request is synchronous. (an disconnect the client.

This process is s.

Define the Acti

Define an Frag oid/support/v4/app/FragmentActivity.html) or Fragment (/reference/a tml) that implements the following interfaces:

ConnectionCa
: Specifie alls when the client is connected or disconnected.

OnConnectionFailedListener
: Specifies a method that Location Serv calls if an error occurs while attempting to connect the client.

For example:

```
public class MainActivity extends Fra gmentActivity implements
        ConnectionCallbacks, OnC     tionFailedListener {
    ...
}
```

Next, define global variables and constants. Defir e constants for the update interval, add a variable for the activity recognition client, and another for th dingIntent (/reference/android/app/PendingIntent.html) that Location Services uses to send updates to your app:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks, OnConnectionFailedListener {
    ...
    // Constants that define the activity detection interval
    public static final int MILLISECONDS_PER_SECOND = 1000;
    public static final int DETECTION_INTERVAL_SECONDS = 20;
    public static final int DETECTION_INTERVAL_MILLISECONDS =
            MILLISECONDS_PER_SECOND * DETECTION_INTERVAL_SECONDS;
    ...
    /*
     * Store the PendingIntent used to send activity recognition events
     * back to the app
     */
    private PendingIntent mActivityRecognitionPendingIntent;
    // Store the current activity recognition client
    private ActivityRecognitionClient mActivityRecognitionClient;
    ...
}
```

In onCreate() (/reference/android/app/Activity.html#onCreate(android.os.Bundle)), instantiate the activity recognition client and the PendingIntent (/reference/android/app/PendingIntent.html):

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks, OnConnectionFailedListener {
    ...
    @Override
    onCreate(Bundle savedInstanceState) {
        ...
        /*
         * Instantiate a new activity recognition client. Since the
         * parent Activity implements the connection listener and
         * connection failure listener, the constructor uses "this"
         * to specify the values of those parameters.
         */
        mActivityRecognitionClient =
                new ActivityRecognitionClient(mContext, this, this);
        /*
         * Create the PendingIntent that Location Services uses
         * to send activity recognition updates back to this app.
         */
        Intent intent = new Intent(
                mContext, ActivityRecognitionIntentService.class);
        /*
         * Return a PendingIntent that starts the IntentService.
         */
        mActivityRecognitionPendingIntent =
                PendingIntent.getService(mContext, 0, intent,
                PendingIntent.FLAG_UPDATE_CURRENT);
        ...
    }
    ...
}
```

Start the request process

Define a method that requests activity recognition updates. In the method, request a connection to Location Services. You can call this method from anywhere in your activity; its purpose is to start the chain of method calls for requesting updates.

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. Set the flag to `true` when you start a request, and then set it to `false` when the request completes.

The following snippet shows how to start a request for updates:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks, OnConnectionFailedListener {
    ...
    // Global constants
    ...
    // Flag that indicates if a request is underway.
    private boolean mInProgress;
    ...
    @Override
    onCreate(Bundle savedInstanceState) {
        ...
        // Start with the request flag set to false
        mInProgress = false;
        ...
    }
    ...
    /**
     * Request activity recognition updates based on the current
     * detection interval.
     *
     */
     public void startUpdates() {
        // Check for Google Play services

        if (!servicesConnected()) {
            return;
        }
        // If a request is not already underway
        if (!mInProgress) {
            // Indicate that a request is in progress
            mInProgress = true;
            // Request a connection to Location Services
            mActivityRecognitionClient.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.
             */
        }
    }
    ...
}
```

Implement onConnected() (/reference/com/google/android/gms/common/GooglePlayServicesClient.ConnectionCallbacks.html#onConnected(android.os.Bundle)). In this method, request activity recognition updates from Location Services. When Location Services finishes connecting to the client and calls onConnected() (/reference/com/google/android/gms/common/GooglePlayServicesClient.ConnectionCallbacks.html#onConnected(android.os.Bundle)), the update request is called immediately:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks, OnConnectionFailedListener {
    ...
    /*
     * Called by Location Services once the location client is connected.
     *
     * Continue by requesting activity updates.
     */
    @Override
    public void onConnected(Bundle dataBundle) {
        /*
         * Request activity recognition updates using the preset
         * detection interval and PendingIntent. This call is
         * synchronous.
         */
        mActivityRecognitionClient.requestActivityUpdates(
                DETECTION_INTERVAL_MILLISECONDS,
                mActivityRecognitionPendingIntent);
        /*
         * Since the preceding call is synchronous, turn off the
         * in progress flag and disconnect the client
         */
        mInProgress = false;
        mActivityRecognitionClient.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/ActivityRecognitionClient.html#disconnect()). To handle this situation, implement onDisconnected() (/reference/com/google/android/gms/common/GooglePlayServicesClient.ConnectionCallbacks.html#onDiscon nected()). 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 {
    ...
    /*
     * Called by Location Services once the activity recognition
     * client is disconnected.
     */
    @Override
    public void onDisconnected() {
        // Turn off the request flag
        mInProgress = false;
        // Delete the client
        mActivityRecognitionClient = 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 {
    ...
    // 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(),
                            "Activity Recognition");
                }
            }
            ...
        }
        ...
}
```

Handle Activity Updates

To handle the Intent (/reference/android/content/Intent.html) that Location Services sends for each update interval, define an IntentService (/reference/android/app/IntentService.html) and its required method onHandleIntent() (/reference/android/app/IntentService.html#onHandleIntent(android.content.Intent)). Location Services sends out activity recognition updates as Intent (/reference/android/content/Intent.html) objects, using the the PendingIntent (/reference/android/app/PendingIntent.html) you provided when you called requestActivityUpdates() (/reference/com/google/android/gms/location/ActivityRecognitionClient.html#requestActivityUpdates(long, android.app.PendingIntent)). Since you provided an explicit intent for the PendingIntent (/reference/android/app/PendingIntent.html), the only component that receives the intent is the IntentService (/reference/android/app/IntentService.html) you're defining.

The following snippets demonstrate how to examine the data in an activity recognition update.

Define an IntentService

Start by defining the class and the required method onHandleIntent() (/reference/android/app/IntentService.html#onHandleIntent(android.content.Intent)):

```
/**
 * Service that receives ActivityRecognition updates. It receives
 * updates in the background, even if the main Activity is not visible.
 */
public class ActivityRecognitionIntentService extends IntentService {
    ...
    /**
     * Called when a new activity detection update is available.
     */
    @Override
    protected void onHandleIntent(Intent intent) {
        ...
    }
    ...
}
```

Next, examine the data in the intent. From the update, you can get a list of possible activities and the probability of each one. The following snippet shows how to get the most probable activity, the confidence level for the activity (the probability that this is the actual activity), and its type:

```
public class ActivityRecognitionIntentService extends IntentService {
    ...
    @Override
    protected void onHandleIntent(Intent intent) {
        ...
        // If the incoming intent contains an update
        if (ActivityRecognitionResult.hasResult(intent)) {
            // Get the update
            ActivityRecognitionResult result =
                    ActivityRecognitionResult.extractResult(intent);
            // Get the most probable activity
            DetectedActivity mostProbableActivity =
                    result.getMostProbableActivity();
            /*
             * Get the probability that this activity is the
             * the user's actual activity
             */
            int confidence = mostProbableActivity.getConfidence();
            /*
             * Get an integer describing the type of activity
             */
            int activityType = mostProbableActivity.getType();
            String activityName = getNameFromType(activityType);
            /*
             * At this point, you have retrieved all the information
             * for the current update. You can display this
             * information to the user in a notification, or
             * send it to an Activity or Service in a broadcast
             * Intent.
             */
            ...
        } else {
            /*
             * This implementation ignores intents that don't contain
             * an activity update. If you wish, you can report them as
             * errors.
             */
        }
        ...
    }
    ...
}
```

The method `getNameFromType()` converts activity types into descriptive strings. In a production app, you should retrieve the strings from resources instead of using fixed values:

```
public class ActivityRecognitionIntentService extends IntentService {
    ...
    /**
     * Map detected activity types to strings
     *@param activityType The detected activity type
     *@return A user-readable name for the type
     */
    private String getNameFromType(int activityType) {
        switch(activityType) {
            case DetectedActivity.IN_VEHICLE:
                return "in_vehicle";
            case DetectedActivity.ON_BICYCLE:
                return "on_bicycle";
            case DetectedActivity.ON_FOOT:
                return "on_foot";
            case DetectedActivity.STILL:
                return "still";
            case DetectedActivity.UNKNOWN:
                return "unknown";
            case DetectedActivity.TILTING:
                return "tilting";
        }
        return "unknown";
    }
    ...
}
```

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.ActivityRecognitionIntentService"
    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 activity update intents are created is described in the section Define the Activity or Fragment ().

Stop Activity Recognition Updates

To stop activity recognition updates, use the same pattern you used to request updates, but call `removeActivityUpdates()` (/reference/com/google/android/gms/location/ActivityRecognitionClient.html#removeActivityUpdates(android.app.PendingIntent)) instead of `requestActivityUpdates()` (/reference/com/google/android/gms/location/ActivityRecognitionClient.html#requestActivityUpdates(long, android.app.PendingIntent)).

Since removing updates uses some of the methods you use to add updates, start by defining request types for the two operations:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks, OnConnectionFailedListener {
    ...
    public enum REQUEST_TYPE = {START, STOP}
    private REQUEST_TYPE mRequestType;
    ...
}
```

Modify the code that starts activity recognition so that it uses the `START` request type:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks, OnConnectionFailedListener {
    ...
    public void startUpdates() {
        // Set the request type to START
        mRequestType = START;
        /*
         * Test for Google Play services after setting the request type.
         * If Google Play services isn't present, the proper request type
         * can be restarted.
         */
        if (!servicesConnected()) {
            return;
        }
        ...
    }
    ...
    public void onConnected(Bundle dataBundle) {
        switch (mRequestType) {
            case START :
                /*
                 * Request activity recognition updates using the
                 * preset detection interval and PendingIntent.
                 * This call is synchronous.
                 */
                mActivityRecognitionClient.requestActivityUpdates(
                        DETECTION_INTERVAL_MILLISECONDS,
                        mActivityRecognitionPendingIntent());
                break;
            ...
        }
        ...
    }
    ...
}
```

Start the process

Define a method that requests a stop to activity recognition updates. In the method, set the request type and then request a connection to Location Services. You can call this method from anywhere in your activity; its purpose is to start the chain of method calls that stop activity updates:

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks, OnConnectionFailedListener {
    ...
    /**
     * Turn off activity recognition updates
     *
     */
    public void stopUpdates() {
        // Set the request type to STOP
        mRequestType = STOP;
        /*
         * 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;
        }
        // If a request is not already underway
        if (!mInProgress) {
            // Indicate that a request is in progress
            mInProgress = true;
            // Request a connection to Location Services
            mActivityRecognitionClient.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.
             */
        }
        ...
    }
    ...
}
```

In onConnected() (/reference/com/google/android/gms/common/GooglePlayServicesClient.ConnectionCallbacks.html#onConnected(android.os.Bundle)), if the request type is STOP, call removeActivityUpdates() (/reference/com/google/android/gms/location/ActivityRecognitionClient.html#removeActivityUpdates(android.app.PendingIntent)). Pass the PendingIntent (/reference/android/app/PendingIntent.html) you used to start updates as the parameter to removeActivityUpdates() (/reference/com/google/android/gms/location/ActivityRecognitionClient.html#removeActivityUpdates(android.app.PendingIntent)):

```
public class MainActivity extends FragmentActivity implements
        ConnectionCallbacks, OnConnectionFailedListener {
    ...
    public void onConnected(Bundle dataBundle) {
        switch (mRequestType) {
            ...
            case STOP :
            mActivityRecognitionClient.removeActivityUpdates(
                    mActivityRecognitionPendingIntent);
            break;

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

You do not have to modify your implementation of onDisconnected() (/reference/com/google/android/gms/common/GooglePlayServicesClient.ConnectionCallbacks.html#onDisconnected()) or onConnectionFailed() (/reference/com/google/android/gms/common/GooglePlayServicesClient.OnConnectionFailedListener.html#onConnectionFailed(com.google.android.gms.common.ConnectionResult)), because these methods do not depend on the request type.

You now have the basic structure of an app that implements activity recognition. You can combine activity recognition with other location-aware features, such as periodic location updates or geofencing, which are described in other lessons in this class.