Google Fit

- Google Fit, part of Google Play Services
- Single cloud storage record for all user’s fitness apps (myfitnesspal), gadgets (fitbit), etc
- Third-party app developers can read-write user’s health data in single Google Fit repository, tied to gmail account (cloud storage)
- User can track health data using multiple devices, apps, store data
Google Fit
Ref:http://en.wikipedia.org/wiki/Google_Fit

● User data not lost if user upgrades, change or loses device

● **Google Fit components:**
  ● **App:** Free from Google, supports fitness tracking, unified view of progress, accessible from multiple devices
  ● **Cloud Storage:** Single repository of user’s fitness data
  ● **API:** Third-party Developers can program app to access, read, write Google Fit record
Google Fit Features

- **Sensors API**: Allows app access raw information from sensors on user’s devices (including smartphones and Android wear devices)
- **Recording API**: Allows app to automate storage of fitness data using subscriptions.
  - Specific data are automatically stored in the background
  - App can access this data on any device user has granted permission to
Google Fit Features

- **History:** App can access user’s fitness history
  - Supports inserting, deleting and querying previously stored fitness data
  - Can also import batch data into Google Fit

- **Bluetooth Low Energy:** Access data directly from Bluetooth, store data from them
  - Apps can find nearby Bluetooth devices and store data from them

![Bluetooth Glucometer](image1.png) ![Bluetooth Weight Scale](image2.png)
Google Fit API
http://en.wikipedia.org/wiki/Google_Fit

- Google Fit API also has API for step counting
- i.e. Low end phones without step counter can use Google Fit’s step counting API
  - Implemented as a Google service
- Also **DetectedActivity** API to detect smartphone user’s current activity
- Currently detects 6 states:
  - In vehicle
  - On Bicycle
  - On Foot
  - Still
  - Tilting
  - Unknown
Using Google Fit
Google Fit API Setup:
Step 1: Create Google Account

- Can use your existing Google account or create a new one for testing
Google Fit API Setup:
Step 2: Get Google Play Services

- Google Play Services: API package and background service
- Allows apps to communicate with Google’s services (e.g. maps, Google+, Google Drive, Google Fit, etc)
- Google Play Services APK
  - Contains individual Google services
  - Runs as background services on Android client that apps interact with
  - Downloaded through Google Play store
- Google updates Play Services often without depending on software updates by phone makers
- Google Play services 7.0 and higher has Google Fit
Google Fit API Setup:

Step 3: Get an Oauth 2.0 Client ID

- Oauth 2.0 is open standard for authorization
- Allows users to log into third party websites using their Microsoft, Google, Facebook or Twitter accounts
- Can get Oauth 2.0 client ID through Google Developers Console
- See: https://developers.google.com/fit/android/get-api-key
- Oauth 2.0 client ID is string of characters. E.g.

```
780816631155-gbvyo1o7r2pn95qc4e19d61io4uh48hl.apps.googleusercontent.com
```
Google Fit API Setup:
Step 4: Configure your Project

- Android Studio is recommended for development
- Create Android Studio project
- Add Google Play services as dependency in **build.gradle** file

```groovy
apply plugin: 'com.android.application'
...

dependencies {
    compile 'com.google.android.gms:play-services-fitness:8.4.0'
}
```

https://developers.google.com/fit/android/get-started
Google Fit API Setup:
Step 5: Connect to the Fitness Service

- Connect to the appropriate fitness service and use it
  - Fitness.SENSORS_API
  - Fitness.RECORDING_API
  - Fitness.HISTORY_API
  - Fitness.SESSIONS_API
  - Fitness.BLE_API
  - Fitness.CONFIG_API
Creating Google Fit Client

**Step 1:** Define variables to help track Google Fit connection

```java
private GoogleApiClient mClient = null;
```
Creating Google Fit Client
Step 2: Connect to Google Fit, Check Permissions

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // This method sets up our custom logger, which will print all log messages to device and logcat
    initializeLogging();

    // When permissions are revoked the app is restarted so onCreate
    if (!checkPermissions()) {
        requestPermissions();
    }
}

@Override
protected void onResume() {
    super.onResume();

    // If user denies permissions then uses Settings to re-enable them, app will start working
    buildFitnessClient();
}
```
Main GoogleFit Client Commands and Callbacks

- **GoogleApiClient.Builder**: Used initially to create GoogleFit client, authenticates user, allows user access Fitness APIs, specifies app scopes
- **Scopes? Read/write permissions to different data types**
- **onConnectionSuspended()**: Called when sensor connection gets lost
- **onConnectionFailed()**: Called when Google Play Services connection fails intentionally
  - Some example reasons for connection failure: User never signed in before, has multiple Google accounts and needs to specify which one to use, etc)
Google Fit Data Types

- Google Fit supports:
  - Instantaneous readings with timestamp (e.g. Current user activity)
  - Aggregate statistics over time interval (e.g. Total calories expended over a time interval)

- 3 Google Fit data types
  - **Public data types**: Standard data types that any app can read and write (e.g. Step count)
  - **Private custom data types**: custom types defined by a specific app. Only that app can read/write this data
  - **Shareable data types**: App developers can submit data types which can be shared after reviewed and approved (E.g. types for Nike Fuel)
## Example Google Fit Public Instantaneous Data Types

<table>
<thead>
<tr>
<th>Data Type Name</th>
<th>Description</th>
<th>Permission</th>
<th>Fields (Format–Unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.google.activity.sample</td>
<td>Instantaneous sample of the current activity.</td>
<td>Activity</td>
<td>activity (int–enum) confidence (float–percent)</td>
</tr>
<tr>
<td>com.google.activity.segment</td>
<td>Continuous time interval of a single activity.</td>
<td>Activity</td>
<td>activity (int–enum)</td>
</tr>
<tr>
<td>(deprecated)com.google.calories.consumed</td>
<td>Total calories consumed over a time interval.</td>
<td>Activity</td>
<td>calories (float–kcal)</td>
</tr>
<tr>
<td>com.google.calories.expended</td>
<td>Total calories expended over a time interval.</td>
<td>Activity</td>
<td>calories (float–kcal)</td>
</tr>
<tr>
<td>com.google.distance.delta</td>
<td>Distance covered since the last reading.</td>
<td>Location</td>
<td>distance (float–meters)</td>
</tr>
<tr>
<td>com.google.heart_rate.bpm</td>
<td>Heart rate in beats per minute.</td>
<td>Body</td>
<td>bpm (float–bpm)</td>
</tr>
<tr>
<td>com.google.height</td>
<td>The user’s height, in meters.</td>
<td>Body</td>
<td>height (float–meters)</td>
</tr>
<tr>
<td><strong>com.google.step_count.cadence</strong></td>
<td>Instantaneous cadence in steps per minute.</td>
<td>Activity</td>
<td>rpm (float–steps/min)</td>
</tr>
<tr>
<td><strong>com.google.step_count.delta</strong></td>
<td>Number of new steps since the last reading.</td>
<td>Activity</td>
<td>steps (int–count)</td>
</tr>
</tbody>
</table>
### Example Google Fit Public Aggregate Data Types

<table>
<thead>
<tr>
<th>Data Type Name</th>
<th>Description</th>
<th>Permission</th>
<th>Fields (Format–Unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.google.activity.summary</td>
<td>Total time and number of segments in a particular activity for a time interval.</td>
<td>Activity</td>
<td>activity (int–enum) duration (int–ms) num_segments (int–count)</td>
</tr>
<tr>
<td>com.google.heart_rate.summary</td>
<td>Average, maximum, and minimum beats per minute for a time interval.</td>
<td>Body</td>
<td>average (float–bpm) max (float–bpm) min (float–bpm)</td>
</tr>
<tr>
<td>com.google.location.bounding_box</td>
<td>A bounding box for the user's location over a time interval.</td>
<td>Location</td>
<td>low_latitude (float–degrees) low_longitude (float–degrees) high_latitude (float–degrees) high_longitude (float–degrees)</td>
</tr>
<tr>
<td>com.google.nutrition.summary</td>
<td>User's nutrition intake during a time interval.</td>
<td>Nutrition</td>
<td>nutrients (Map&lt;String, float&gt;–calories/grams/IU) meal_type (int–enum) food_item (String–n/a)</td>
</tr>
<tr>
<td>com.google.power.summary</td>
<td>Average, maximum, and minimum power generated while performing an activity.</td>
<td>Activity</td>
<td>average (float–watts) max (float–watts) min (float–watts)</td>
</tr>
<tr>
<td>com.google.speed.summary</td>
<td>Average, maximum, and minimum speed over ground over a time interval.</td>
<td>Location</td>
<td>average (float–m/s) max (float–m/s) min (float–m/s)</td>
</tr>
<tr>
<td>com.google.weight.summary</td>
<td>Average, maximum, and minimum weight over a time interval.</td>
<td>Body</td>
<td>average (float–kg) max (float–kg) min (float–kg)</td>
</tr>
</tbody>
</table>
### Google Fit Data Scopes

- Scopes are strings that specify:
  - Types of data app can access
  - Level of access (Read/write permissions)

- App requests a scope of access during initial connection, access data if permission received

<table>
<thead>
<tr>
<th>Permission</th>
<th>Scope</th>
<th>Type of Access</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td>FITNESS_ACTIVITY_READ_READ</td>
<td>Read</td>
<td>com.google.activity.sample, com.google.activity.segment, com.google.activity.summary, (deprecated) com.google.calories.consumed, com.google.calories.expended, com.google.cycling.pedaling.cadence, com.google.power.sample</td>
</tr>
<tr>
<td></td>
<td>FITNESS_ACTIVITY_READ_WRITE</td>
<td>Read and Write</td>
<td>com.google.step_count.cadence, com.google.step_count.delta, com.google.activity.exercise</td>
</tr>
<tr>
<td>Body</td>
<td>FITNESS_BODY_READ</td>
<td>Read</td>
<td>com.google.heart_rate.bpm, com.google.heart_rate.summary, com.google.height, com.google.weight, com.google.weight.summary</td>
</tr>
<tr>
<td></td>
<td>FITNESS_BODY_READ_WRITE</td>
<td>Read and Write</td>
<td></td>
</tr>
</tbody>
</table>
## Google Fit Data Scopes

<table>
<thead>
<tr>
<th>Permission</th>
<th>Scope</th>
<th>Type of Access</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>FITNESS_LOCATION_READ</td>
<td>Read</td>
<td>com.google.cycling.wheel_revolution.cumulative</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>com.google.cycling.wheel.revolutions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>com.google.distance.delta</td>
</tr>
<tr>
<td></td>
<td>FITNESS_LOCATION_READ_WRITE</td>
<td>Read and Write</td>
<td>com.google.location.sample</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>com.google.location.bounding_box</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>com.google.speed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>com.google.speed.summary</td>
</tr>
<tr>
<td>Nutrition</td>
<td>FITNESS_NUTRITION_READ</td>
<td>Read</td>
<td>com.google.nutrition.item</td>
</tr>
<tr>
<td></td>
<td>FITNESS_NUTRITION_READ_WRITE</td>
<td>Read and Write</td>
<td>com.google.nutrition.summary</td>
</tr>
</tbody>
</table>
Creating Google Fit Client
Step 2: Connect to Google Fit, Check Permissions

```java
private void buildFitnessClient() {
    if (mClient == null && checkPermissions()) {
        mClient = new GoogleApiClient.Builder(this)
                .addApi(Fitness.SENSORS_API)
                .addScope(new Scope(Scopes.FITNESS_LOCATION_READ))
                .addConnectionCallbacks(
                        new GoogleApiClient.ConnectionCallbacks() {
                            @Override
                            public void onConnected(Bundle bundle) {
                                Log.i(TAG, "Connected!!!");
                                // Now you can make calls to the Fitness APIs.
                                findFitnessDataSources();
                            }
                        })
                .build();
    }
}
```
Creating Google Fit Client
Step 2: Connect to Google Fit, Check Permissions

```java
@Override
public void onConnectionSuspended(int i) {
// Called if sensor connection lost
    // If your connection to the sensor gets lost at some point,
    // you'll be able to determine the reason and react to it here.
    if (i == ConnectionCallbacks.CAUSE_NETWORK_LOST) {
        Log.i(TAG, "Connection lost. Cause: Network Lost.");
    } else if (i == ConnectionCallbacks.CAUSE_SERVICE_DISCONNECTED) {
        Log.i(TAG, "Connection lost. Reason: Service Disconnected");
    }
}
```
Creating Google Fit Client
Step 2: Connect to Google Fit, Check Permissions

.enableAutoManage(this, 0, new GoogleApiClient.OnConnectionFailedListener() {
    @Override
    public void onConnectionFailed(ConnectionResult result) {
        Log.i(TAG, "Google Play services connection failed. Cause: " +
            result.toString());
        Snackbar.make(
            MainActivity.this.findViewById(R.id.main_activity_view),
            "Exception while connecting to Google Play services: " +
            result.getErrorMessage(),
            Snackbar.LENGTH_INDEFINITE).show();
    }
}).build();
Activity Recognition Using Google Fit
Activity Recognition Using Google Fit
Ref: How to Recognize User Activity with Activity Recognition by Paul Trebilcox-Ruiz on Tutsplus.com tutorials

● Google Fit can:
  ● Recognize user’s current activity (Running, walking, in a vehicle or still)
  ● Why? E.g. If user is driving, don’t send notifications
  ● Track user’s steps

● Project Setup similar to previously described case:
  ● Create Android Studio project with blank Activity (minimum SDK 14)
  ● In build.gradle file, define latest Google Play services (8.4) as dependency
Activity Recognition Using Google Fit
Ref: How to Recognize User Activity with Activity Recognition by Paul Trebilcox-Ruiz on Tutsplus.com tutorials

- Create new class `ActivityRecognizedService` which extends `IntentService`
- Throughout user’s day, GooglePlay sends user’s activity to this `IntentService`
- Need to program this Intent to handle incoming user activity

```java
public class ActivityRecognizedService extends IntentService {

    public ActivityRecognizedService() {
        super("ActivityRecognizedService");
    }

    public ActivityRecognizedService(String name) {
        super(name);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
    }
}
```
Activity Recognition Using Google Fit

Ref: How to Recognize User Activity with Activity Recognition by Paul Trebilcox-Ruiz on Tutsplus.com tutorials

- Modify **AndroidManifest.xml** to
  - Declare **ActivityRecognizedService**
  - Add com.google.android.gms.permission.ACTIVITY_RECOGNITION permission

```xml
    <?xml version="1.0" encoding="utf-8"?>
    <manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.tutsplus.activityrecognition">
        <uses-permission
          android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION" />

        <application
          android:icon="@mipmap/ic_launcher"
          android:label="@string/app_name"
          android:theme="@style/AppTheme">
          <activity android:name=".MainActivity">
            <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
          </activity>
          <service android:name=".ActivityRecognizedService" />
        </application>
    </manifest>
```
Project Setup

- To connect to Google Play Services, provide `GoogleApiClient` variable type and implement callbacks.

```java
public class MainActivity extends AppCompatActivity implements GoogleApiClient.ConnectionCallbacks,
          GoogleApiClient.OnConnectionFailedListener {

    public GoogleApiClient mApiClient;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public void onConnected(@Nullable Bundle bundle) {
    }

    @Override
    public void onConnectionSuspended(int i) {
    }

    @Override
    public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {
    }
}
**Requesting Activity Recognition**

- In `onCreate`, initialize client and connect to Google Play Services

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mApiClient = new GoogleApiClient.Builder(this)
        .addApi(ActivityRecognition.API)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .build();

    mApiClient.connect();
}
```
Requesting Activity Recognition

- Once `GoogleApiClient` has connected, `onConnected()` instance is called.
- Need to create a `PendingIntent` that goes to our `IntentService`.
- Also set how often API should check user’s activity in milliseconds.

```java
@Override
public void onConnected(@Nullable Bundle bundle) {
    Intent intent = new Intent(this, ActivityRecognizedService.class);
    PendingIntent pendingIntent = PendingIntent.getService(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);
    ActivityRecognition.ActivityRecognitionApi.requestActivityUpdates(mApiClient, 3000, pendingIntent);
}
```
Handling Activity Recognition

- Our app now needs to attempt to recognize the user’s activity every 3 seconds, send data to `ActivityRecognizedService`

- In `onHandleIntent()` method of `ActivityRecognizedService`
  - Validate that received intent contains activity recognition data
  - If so, extract `ActivityRecognitionResult` from the Intent
  - Retrieve list of possible activities by calling `getProbableActivities()` on `ActivityRecognitionResult` object

```java
@Override
protected void onHandleIntent(Intent intent) {
    if(ActivityRecognitionResult.hasResult(intent)) {
        ActivityRecognitionResult result = ActivityRecognitionResult.extractResult(intent);
        handleDetectedActivities(result.getProbableActivities());
    }
}
```
Handling Activity Recognition

- Simply log each detected activity and display how confident Google Play services is that user is performing this activity.

```java
private void handleDetectedActivities(List<DetectedActivity> probableActivities) {
    for (DetectedActivity activity : probableActivities) {
        switch (activity.getType()) {
            case DetectedActivity.IN_VEHICLE: {
                Log.e("ActivityRecognition", "In Vehicle: " + activity.getConfidence());
                break;
            }
            case DetectedActivity.ON_BICYCLE: {
                Log.e("ActivityRecognition", "On Bicycle: " + activity.getConfidence());
                break;
            }
            case DetectedActivity.ON_FOOT: {
                Log.e("ActivityRecognition", "On Foot: " + activity.getConfidence());
                break;
            }
            case DetectedActivity.RUNNING: {
                Log.e("ActivityRecognition", "Running: " + activity.getConfidence());
                break;
            }
            case DetectedActivity.STILL: {
                Log.e("ActivityRecognition", "Still: " + activity.getConfidence());
                break;
            }
            case DetectedActivity.TILTING: {
                Log.e("ActivityRecognition", "Tilting: " + activity.getConfidence());
                break;
            }
        }
    }
```
Handling Activity Recognition

- If confidence is > 75, activity detection is probably accurate

```java
    case DetectedActivity.WALKING: {
        Log.e( "ActivityRecogition", "Walking: " + activity.getConfidence() );
        if( activity.getConfidence() >= 75 ) {
            NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
            builder.setContentText( "Are you walking?" );
            builder.setSmallIcon( R.mipmap.ic_launcher );
            builder.setContentTitle( getString( R.string.app_name ) );
            NotificationManagerCompat.from(this).notify(0, builder.build());
        }
        break;
    }
    case DetectedActivity.UNKNOWN: {
        Log.e( "ActivityRecogition", "Unknown: " + activity.getConfidence() );
        break;
    }
```
Sample Output of Program

- Sample displayed on development console

<table>
<thead>
<tr>
<th></th>
<th>E/ActivityRecognition: On Foot: 92</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>E/ActivityRecognition: Running: 87</td>
</tr>
<tr>
<td>3</td>
<td>E/ActivityRecognition: On Bicycle: 8</td>
</tr>
<tr>
<td>4</td>
<td>E/ActivityRecognition: Walking: 5</td>
</tr>
</tbody>
</table>

- Or provided as notification to user

- Full code at: https://github.com/tutsplus/Android-ActivityRecognition